Introduction to Artificial Intelligence

丁尧相
浙江大学

Fall & Winter 2022
Week 6

Announcements

- We will release Problem Set 2.1 after this lecture.
- No lab class this Wednesday.
- While the first lab project will be out!

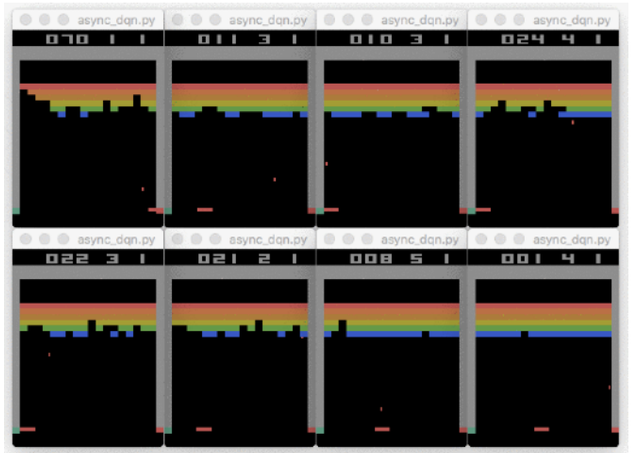
Knowledge Reasoning: I

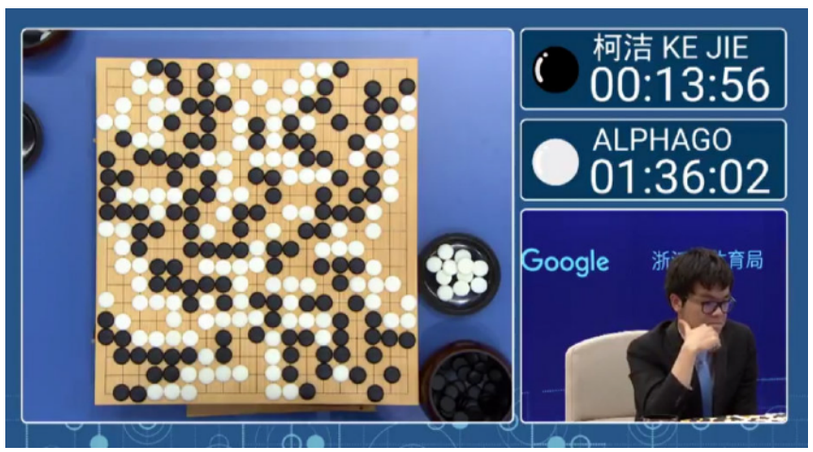
- Logic Reasoning
- Propositional Logic
  - Search-Based Inference
  - Resolution-Based Inference
  - Forward and Backward Chaining
- First-Order Logic
- Take-Home Messages

Knowledge Reasoning: I

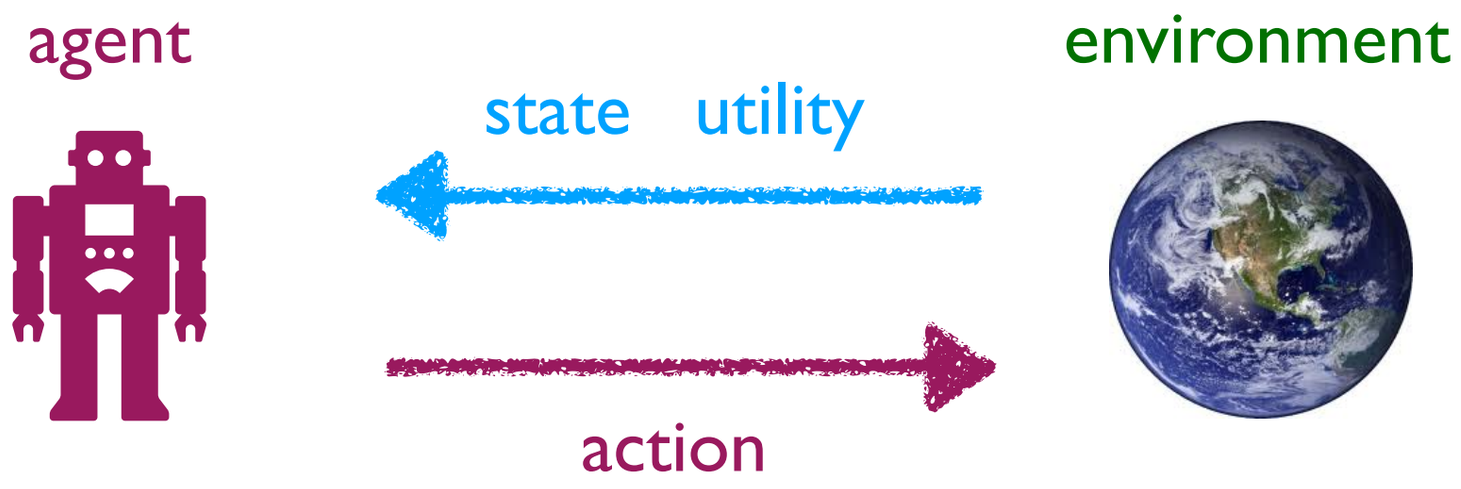
- Logic Reasoning
- Propositional Logic
  - Search-Based Inference
  - Resolution-Based Inference
  - Forward and Backward Chaining
- First-Order Logic
- Take-Home Messages

Decision Making



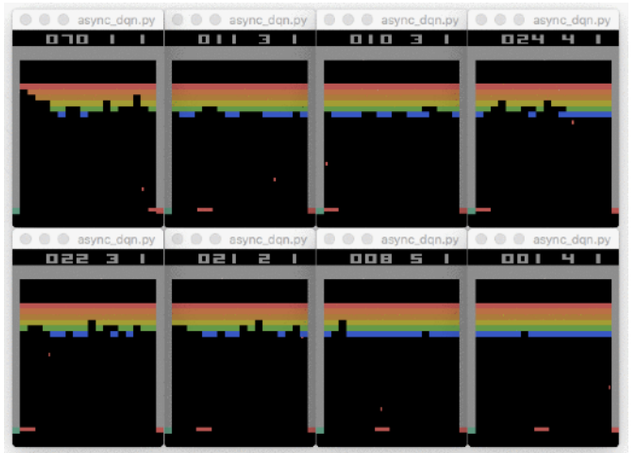


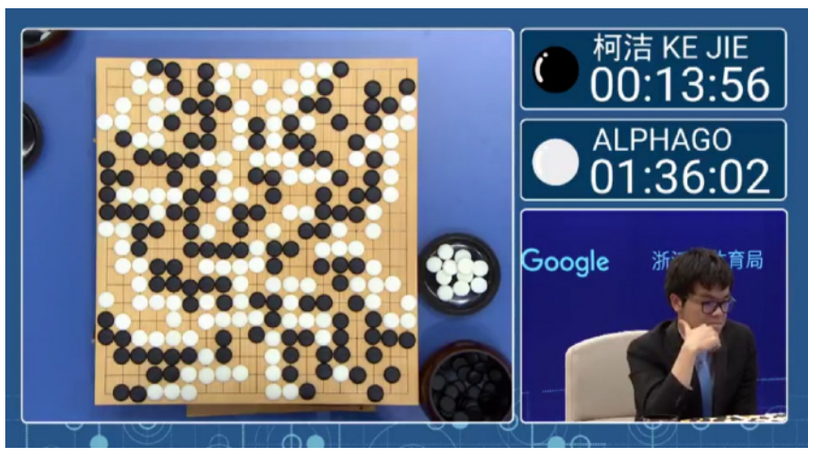
- Conduct action in any state of an environment.



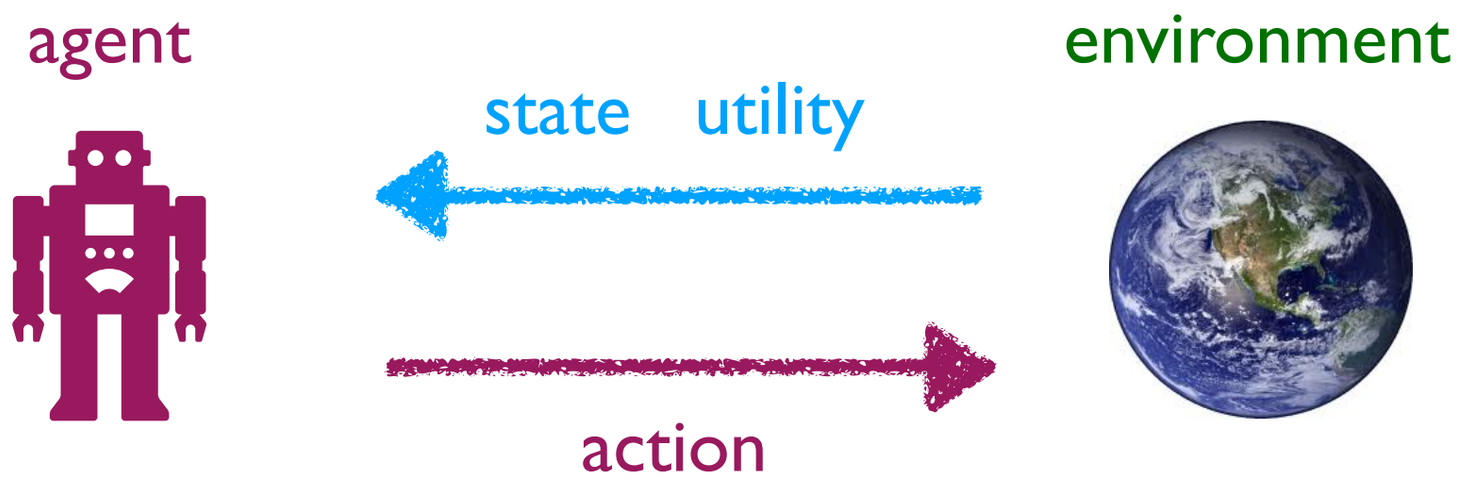
In most problems, the agent needs to do a sequence of actions w.r.t. a sequence of states.

Decision Making





- Conduct action in any state of an environment.



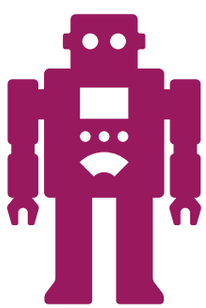
In most problems, the agent needs to do a sequence of actions w.r.t. a sequence of states.

Internal vs. External Model

Since the agent cannot fully know the external model,
it should build an internal model itself for decision making.

internal model

agent



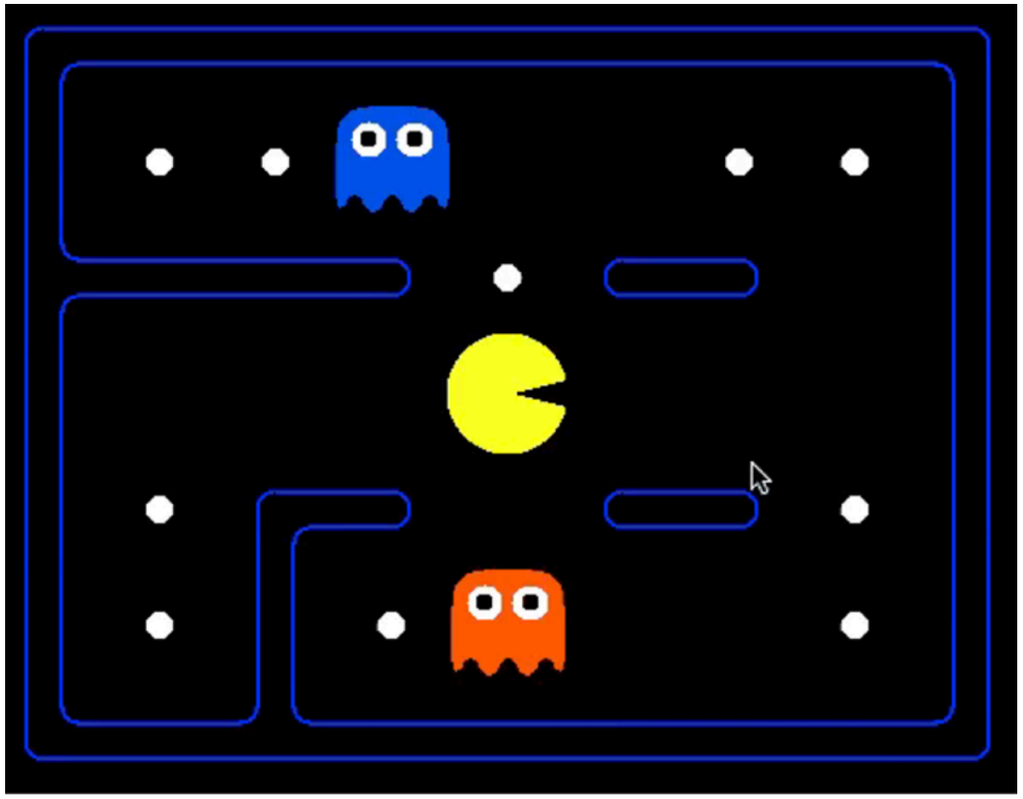
partial state

utility

action

environment
external model

Knowledge in Pacman



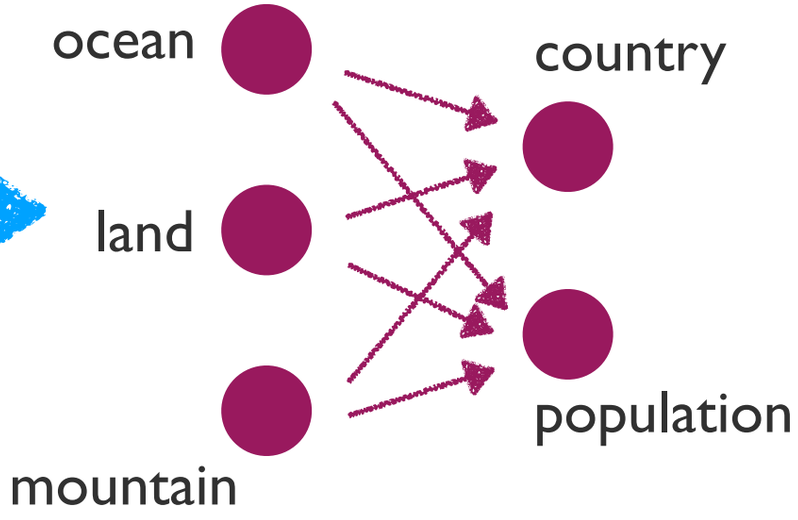
know only the positions
vs.
know the distance to the ghosts
vs.
know the high-level strategy of the ghosts

The search agent knows the external model,
but it can make no changes or abstractions when the model is primitive.
The knowledge-based agents can benefit from the internal model
not just by covering the external model.

Knowledge in AI Systems

longitude and latitude

ocean, land, mountain

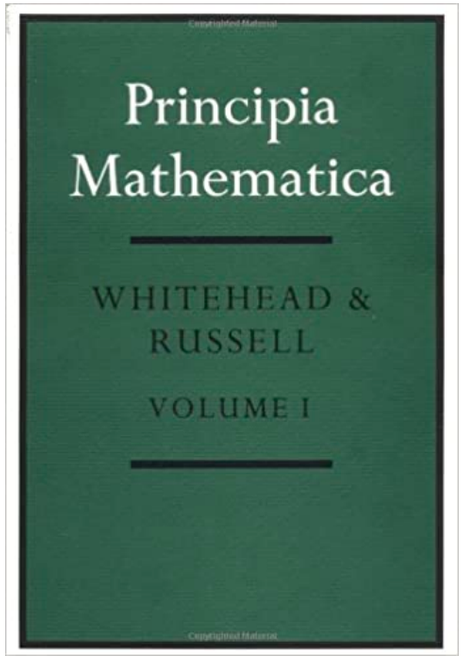


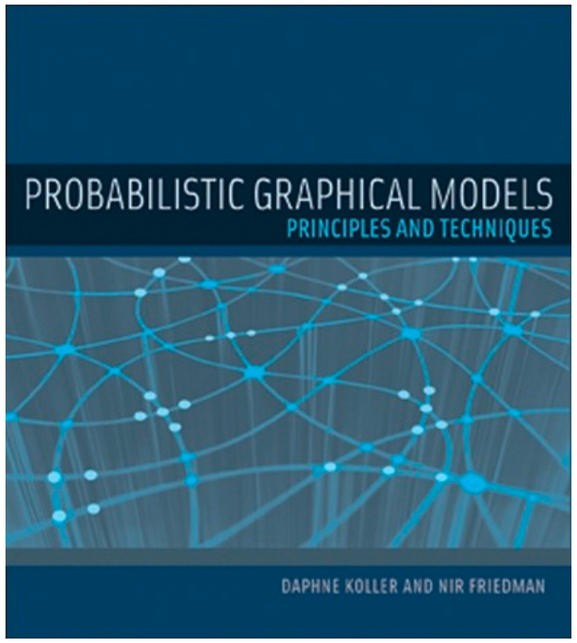
- Turn primitive external states into meaningful internal states.
- Reason about most useful states for decision making.
- Capture internal relationships among factors of decision making.

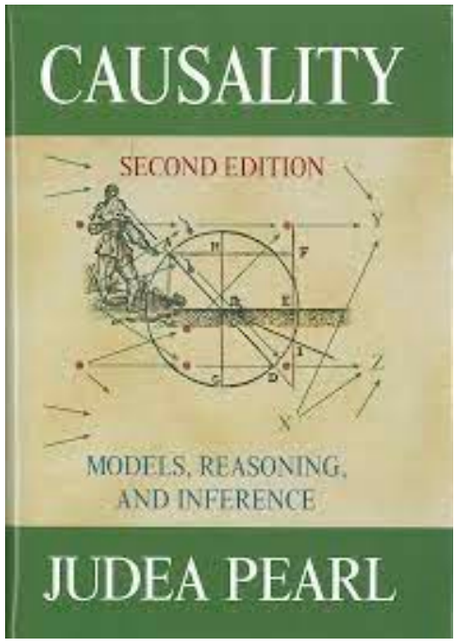
These reasoning rules are called knowledges in an AI system.

Knowledge Reasoning Systems

- Logic reasoning
- Probabilistic reasoning
- Causal reasoning



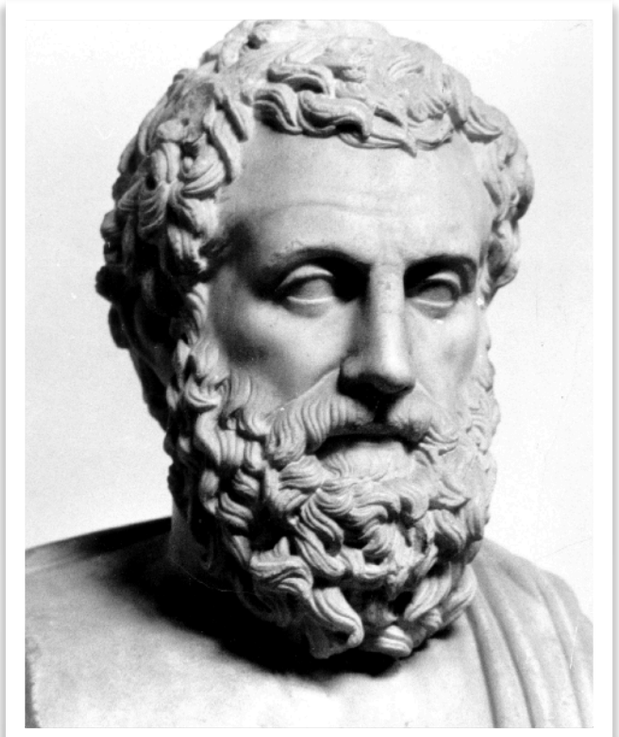


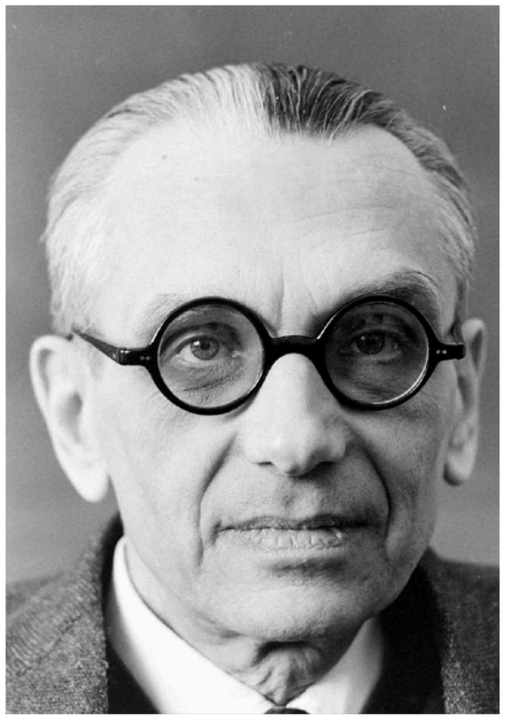


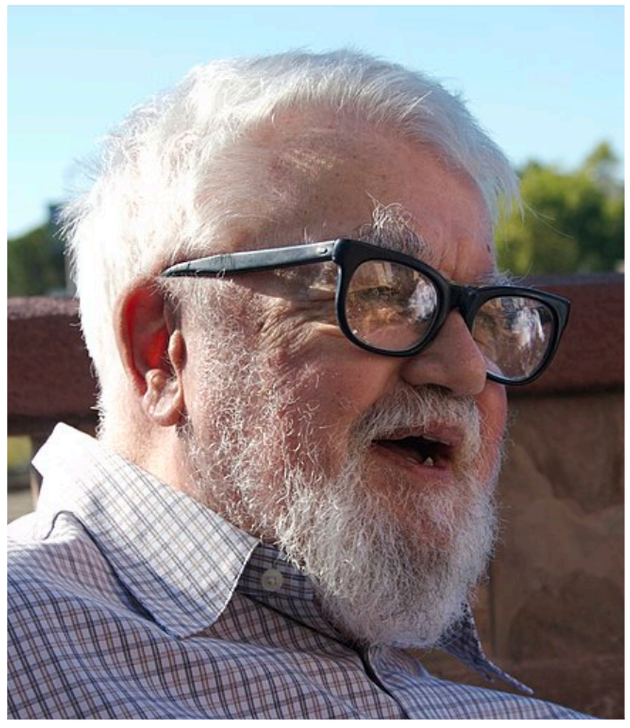
Currently we assume that the agents can access to
a knowledge base (facts) and a reasoning rule system
but cannot change them.
In some sense, the agents just use knowledge but cannot obtain or increase.

Logic Reasoning Systems

- Handling decision problems (true/false arguments).
- Handling discrete and (not exactly) deterministic world.

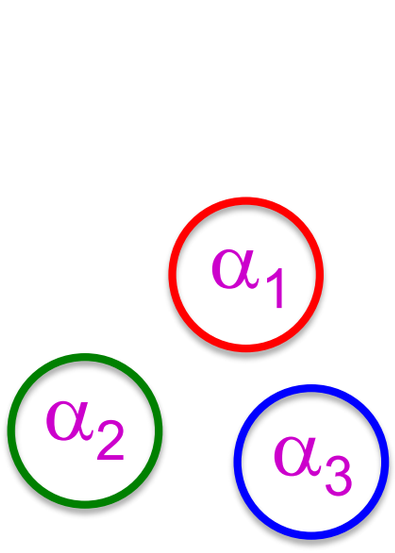






Building Blocks of Logic Systems: Syntax & Semantics

- ***Syntax***: What sentences are allowed?
- ***Semantics***:
  - What are the ***possible worlds***?
  - Which sentences are ***true*** in which worlds? (i.e., ***definition*** of truth)



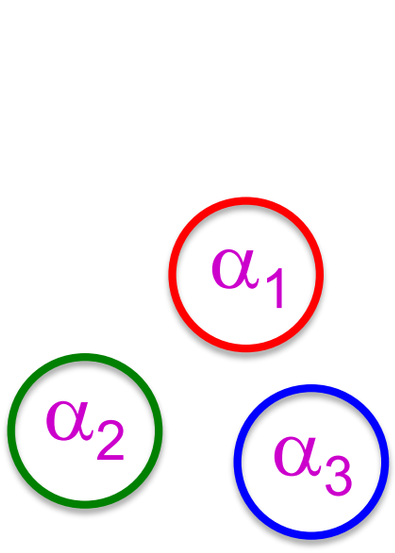
Syntaxland

Semanticsland

Slide courtesy: Stuart Russell & Sergey Levine

Building Blocks of Logic Systems: Syntax & Semantics

- ***Syntax***: What sentences are allowed?
- ***Semantics***:
  - What are the ***possible worlds***? models
  - Which sentences are ***true*** in which worlds? (i.e., ***definition*** of truth)



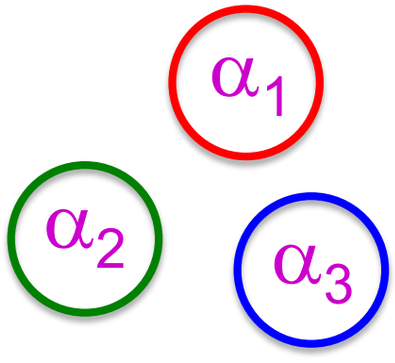
Syntaxland

Semanticsland

Slide courtesy: Stuart Russell & Sergey Levine

Building Blocks of Logic Systems: Syntax & Semantics

- ***Syntax***: What sentences are allowed?
- ***Semantics***:
  - What are the ***possible worlds***? models
  - Which sentences are ***true*** in which worlds? (i.e., ***definition*** of truth)



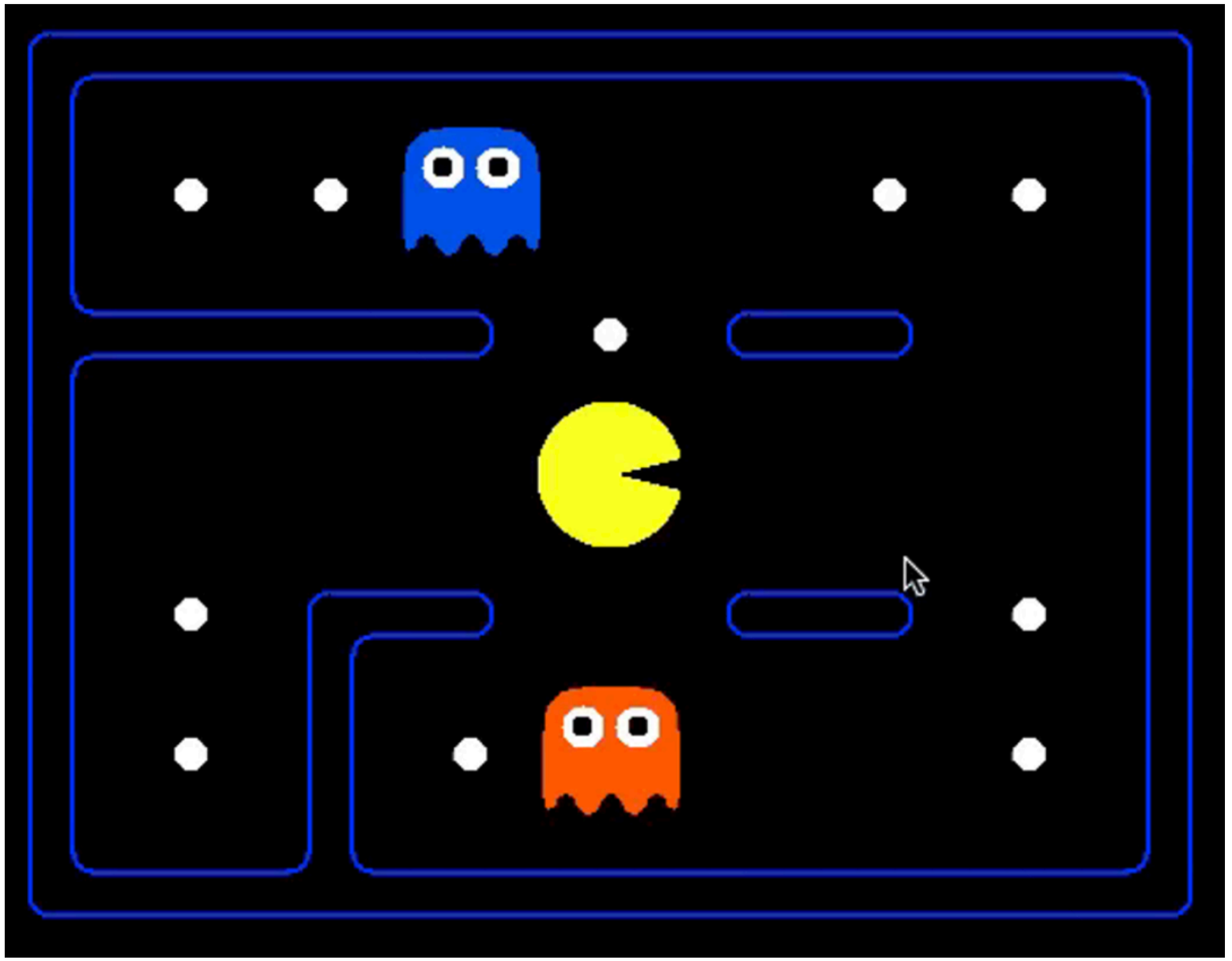
Syntaxland

Semanticsland

Example: 1+1=2

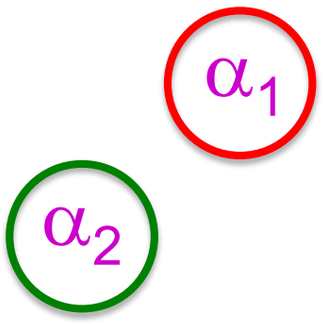
Slide courtesy: Stuart Russell & Sergey Levine

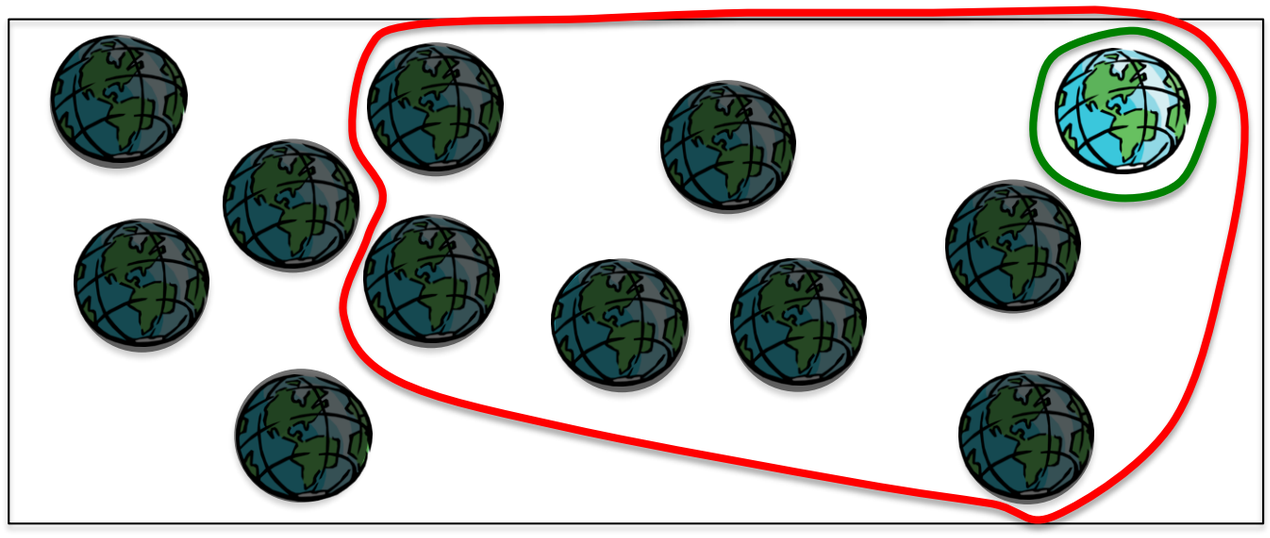
The Pacman Example



Logic Inference: Entailment

- ***Entailment***: $\alpha \models \beta$ ("α entails β" or "β follows from α") iff in every world where α is true, β is also true
  - I.e., the α-worlds are a subset of the β-worlds [***models***(α) $\subseteq$ ***models***(β)]
- In the example, $\alpha_2 \models \alpha_1$
- (Say α_2 is $\neg Q \wedge R \wedge S \wedge W$
  α_1 is $\neg Q$)

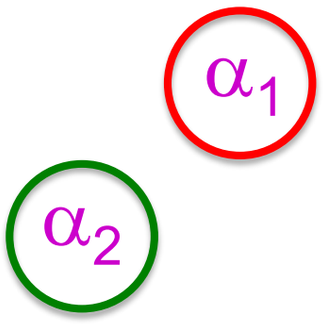


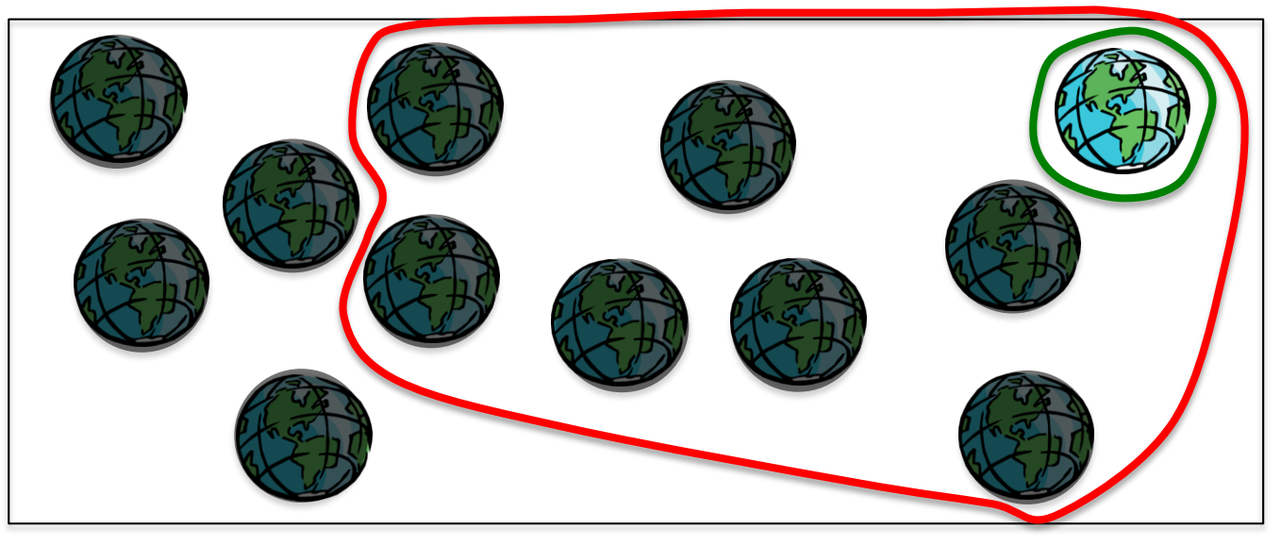


Slide courtesy: Stuart Russell & Sergey Levine

Logic Inference: Entailment

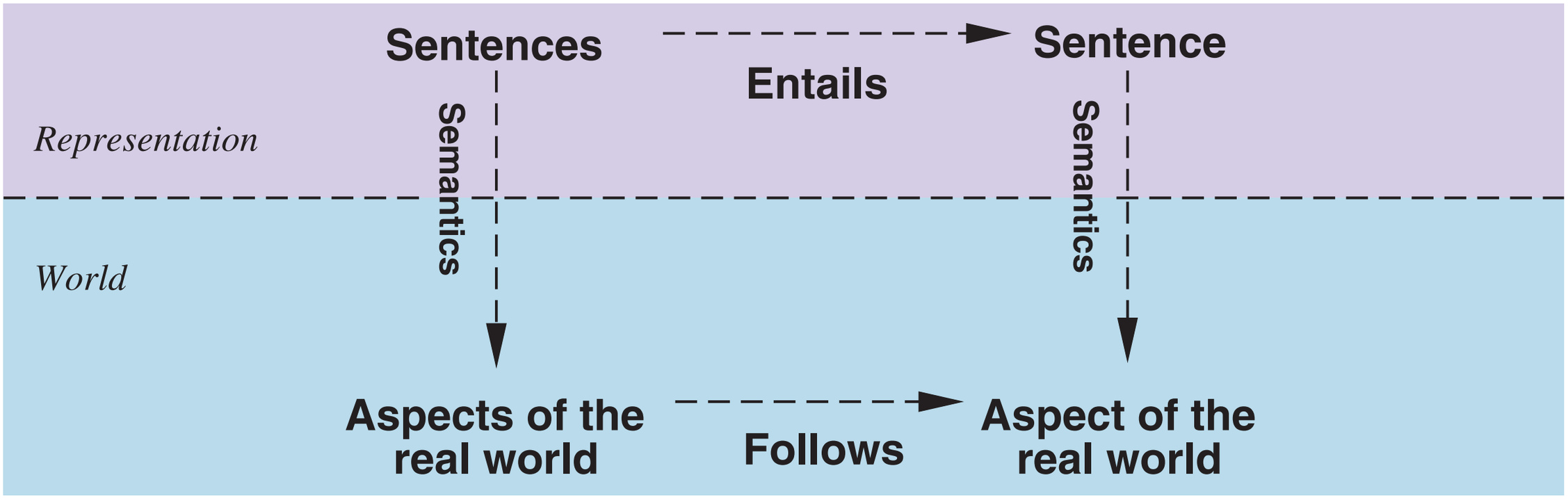
- ***Entailment***: $\alpha \models \beta$ ("α entails β" or "β follows from α") iff in every world where α is true, β is also true
  - I.e., the α-worlds are a subset of the β-worlds [***models***(α) $\subseteq$ ***models***(β)]
- In the example, $\alpha_2 \models \alpha_1$
- (Say α_2 is $\neg Q \wedge R \wedge S \wedge W$
  α_1 is $\neg Q$)





Slide courtesy: Stuart Russell & Sergey Levine

Syntax vs. Semantics (Cont.)



Semantics need to have groundings in the real world.

Knowledge Reasoning: I

- Logic Reasoning
- Propositional Logic
  - Search-Based Inference
  - Resolution-Based Inference
  - Forward and Backward Chaining
- First-Order Logic
- Take-Home Messages

Propositional Logic: Syntax

- Given propositional true/false symbols $\{X_1, X_2, X_3, ..., X_n\}$
- The following logic sentences are legal:
  - All X_i
  - If a is legal, then $\neg a$ is legal
  - if a and b are legal, then $a \wedge b$ is legal
  - if a and b are legal, then $a \vee b$ is legal
  - if a and b are legal, then $a \Rightarrow b$ is legal
  - if a and b are legal, then $a \Leftrightarrow b$ is legal

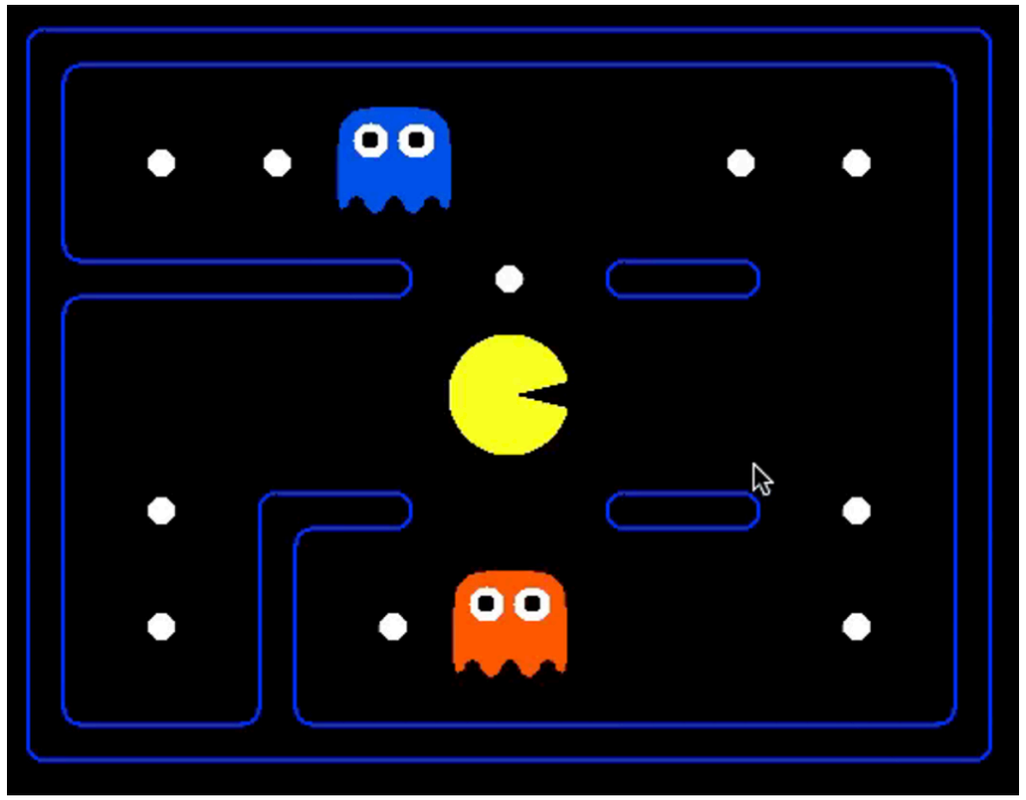
We have not defined the rules of inference!

Propositional Logic: Semantics

- For any logical sentences P, Q and any model m
  - $\neg P$ is true iff P is false in m
  - $P \wedge Q$ is true iff both P and Q is true in m
  - $P \vee Q$ is true iff either P or Q is true in m
  - $P \Rightarrow Q$ is true unless P is true but Q is false in m
  - $P \Leftrightarrow Q$ is true iff P and Q are both true or false in m

Question: What is the role of the model?

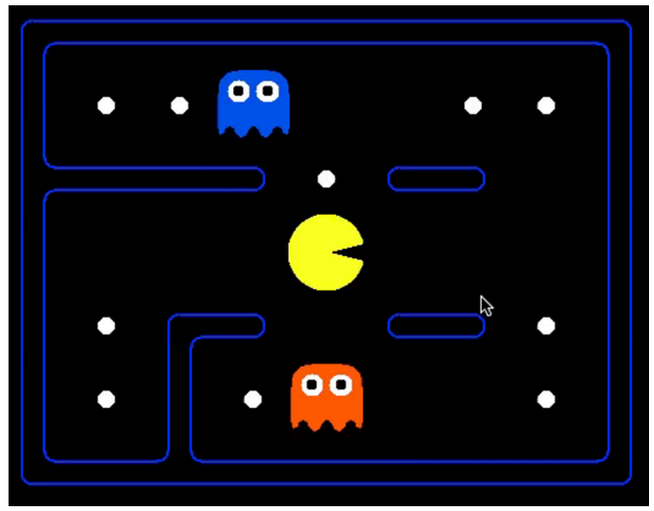
The Inference Procedure



- Agent action: $A_{left}, A_{right}, A_{up}, A_{down}$
- Ghost 1 position: $G_{1,left}, G_{1,right}, G_{1,up}, G_{1,down}$
- Ghost 2 position: $G_{2,left}, G_{2,right}, G_{2,up}, G_{2,down}$

The Inference Procedure

- Agent action: $A_{left}, A_{right}, A_{up}, A_{down}$



- Ghost 1 position: $G_{1,left}, G_{1,right}, G_{1,up}, G_{1,down}$
- Ghost 2 position: $G_{2,left}, G_{2,right}, G_{2,up}, G_{2,down}$
- Given knowledge base (KB):
  - $G_{1,left} \Rightarrow \neg A_{left}$ $G_{1,right} \Rightarrow \neg A_{right}$ $G_{1,up} \Rightarrow \neg A_{up}$
  - $G_{2,left} \Rightarrow \neg A_{left}$
- Inference: whether $G_{1,left} \vee G_{2,left} \Rightarrow \neg A_{left}$

From the definition of entailment:
The simplest method is to enumerate all worlds satisfying the variables to see whether the sentence always holds.
Too expensive and impossible when infinite. Better ideas?

Logic Inference & SAT

$a \models b$

- $a \Rightarrow b$ is equivalent to $\neg a \vee b$, whose negative is $a \wedge \neg b$
- Thus we can equivalently show no world satisfies $a \wedge \neg b$
- Furthermore, using the idea of proof by contradiction, we can try to find a world which satisfies $a \wedge \neg b$, if we are failed, then we have proved $a \Rightarrow b$.

Determining the satisfiability of sentences in proposition logic:
The SAT problem.
The first problem proved to be NP-complete.

Inference Approaches

- A proof is a ***demonstration*** of entailment between α and β
- Method 1: ***model-checking***
  - For every possible world, if α is true make sure that is β true too
  - OK for propositional logic (finitely many worlds); not easy for first-order logic
- Method 2: ***theorem-proving***
  - Search for a sequence of proof steps (applications of ***inference rules***) leading from α to β
  - E.g., from $P \wedge (P \Rightarrow Q)$, infer Q by ***Modus Ponens***
- ***Sound*** algorithm: everything it claims to prove is in fact entailed
- ***Complete*** algorithm: every that is entailed can be proved

Notions of different completeness:
Gödel's completeness and incompleteness theorems.

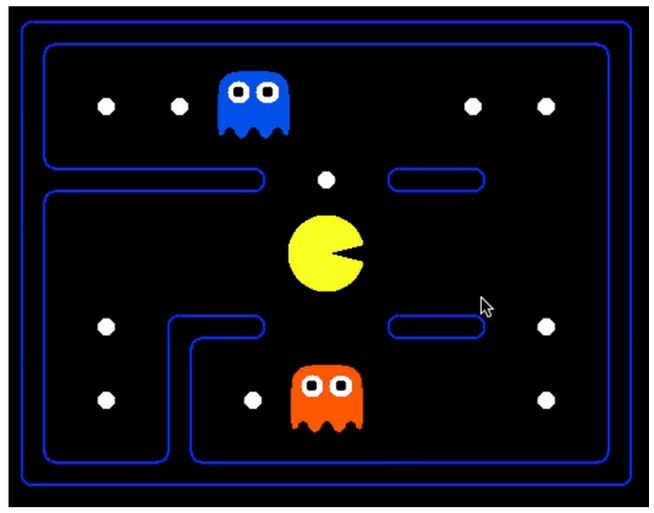
Slide courtesy: Stuart Russell & Sergey Levine

Knowledge Reasoning: I

- Logic Reasoning
- Propositional Logic
  - Search-Based Inference
  - Resolution-Based Inference
  - Forward and Backward Chaining
- First-Order Logic
- Take-Home Messages

Theorem Proving in Pacman

- Agent action: $A_{left}, A_{right}, A_{up}, A_{down}$



- Ghost 1 position: $G_{1,left}, G_{1,right}, G_{1,up}, G_{1,down}$
- Ghost 2 position: $G_{2,left}, G_{2,right}, G_{2,up}, G_{2,down}$
- Given knowledge base (KB):
  - $G_{1,left} \Rightarrow \neg A_{left}$ $\quad G_{1,right} \Rightarrow \neg A_{right}$ $\quad G_{1,up} \Rightarrow \neg A_{up}$
  - $G_{2,left} \Rightarrow \neg A_{left}$
- Inference: whether $G_{1,left} \vee G_{2,left} \Rightarrow \neg A_{left}$

Applying logic rules in sequence:

$$(G_{1,left} \Rightarrow \neg A_{left}) \equiv \neg G_{1,left} \vee \neg A_{left}$$

$$(G_{2,left} \Rightarrow \neg A_{left}) \equiv \neg G_{2,left} \vee \neg A_{left}$$

$$(\neg G_{1,left} \vee A_{left}) \wedge (\neg G_{2,left} \vee A_{left}) \equiv \neg (G_{1,left} \vee G_{2,left}) \vee \neg A_{left}$$

Theorem Proving by Search

- Initial State: the initial knowledge base.
- State: all the legal sentences
- Action: all the legal inference rules
- Transition: the results of the rules
- Goal: the target sentence to prove
- Cost: a step of inference

Can apply any search problems in lecture 2!
Need to pre-define inference rules. Not guaranteed to be complete if misspecified.

Knowledge Reasoning: I

- Logic Reasoning
- Propositional Logic
  - Search-Based Inference
  - Resolution-Based Inference
  - Forward and Backward Chaining
- First-Order Logic
- Take-Home Messages

The Resolution Rule

- Unit resolution rule:

$$(X_1 \vee X_2 \vee ... \vee X_i \vee ... \vee X_k) \wedge \neg X_i \equiv (X_1 \vee X_2 \vee ... \vee X_{i-1} \vee X_{i+1} \vee ... \vee X_k)$$

- Full resolution rule:

$$(X_1 \vee ... \vee X_k) \wedge (\neg X_i \vee Y_1 \vee ... \vee Y_m) \equiv (X_1 \vee ... \vee X_{i-1} \vee X_{i+1} \vee ... \vee X_k \vee Y_1 \vee ... \vee Y_m)$$

In the conjunction of two disjunctions,
the complementary variables can be eliminated.

The Resolution Rule

- Unit resolution rule:

$$(X_1 \vee X_2 \vee ... \vee X_i \vee ... \vee X_k) \wedge \neg X_i \equiv (X_1 \vee X_2 \vee ... \vee X_{i-1} \vee X_{i+1} \vee ... \vee X_k)$$

- Full resolution rule:

$$(X_1 \vee ... \vee X_k) \wedge (\neg X_i \vee Y_1 \vee ... \vee Y_m) \equiv (X_1 \vee ... \vee X_{i-1} \vee X_{i+1} \vee ... \vee X_k \vee Y_1 \vee ... \vee Y_m)$$

In the conjunction of two disjunctions,
the complementary variables can be eliminated.

The Resolution Rule

- Unit resolution rule:

$$(X_1 \vee X_2 \vee ... \vee X_i \vee ... \vee X_k) \wedge \neg X_i \equiv (X_1 \vee X_2 \vee ... \vee X_{i-1} \vee X_{i+1} \vee ... \vee X_k)$$

- Full resolution rule:

$$(X_1 \vee ... \vee X_k) \wedge (\neg X_i \vee Y_1 \vee ... \vee Y_m) \equiv (X_1 \vee ... \vee X_{i-1} \vee X_{i+1} \vee ... \vee X_k \vee Y_1 \vee ... \vee Y_m)$$

In the conjunction of two disjunctions,
the complementary variables can be eliminated.

$$(P \vee \neg Q \vee R) \wedge (\neg P \vee Q) \equiv ?$$

The Resolution Rule

- Unit resolution rule:

$$(X_1 \vee X_2 \vee ... \vee X_i \vee ... \vee X_k) \wedge \neg X_i \equiv (X_1 \vee X_2 \vee ... \vee X_{i-1} \vee X_{i+1} \vee ... \vee X_k)$$

- Full resolution rule:

$$(X_1 \vee ... \vee X_k) \wedge (\neg X_i \vee Y_1 \vee ... \vee Y_m) \equiv (X_1 \vee ... \vee X_{i-1} \vee X_{i+1} \vee ... \vee X_k \vee Y_1 \vee ... \vee Y_m)$$

In the conjunction of two disjunctions,
the complementary variables can be eliminated.

$$(P \vee \neg Q \vee R) \wedge (\neg P \vee Q) \equiv ?$$

$$R?$$
$$\neg Q \vee Q \vee R?$$

The Resolution Rule

- Unit resolution rule:

$$(X_1 \vee X_2 \vee ... \vee X_i \vee ... \vee X_k) \wedge \neg X_i \equiv (X_1 \vee X_2 \vee ... \vee X_{i-1} \vee X_{i+1} \vee ... \vee X_k)$$

- Full resolution rule:

$$(X_1 \vee ... \vee X_k) \wedge (\neg X_i \vee Y_1 \vee ... \vee Y_m) \equiv (X_1 \vee ... \vee X_{i-1} \vee X_{i+1} \vee ... \vee X_k \vee Y_1 \vee ... \vee Y_m)$$

In the conjunction of two disjunctions,
the complementary variables can be eliminated.

$$(P \vee \neg Q \vee R) \wedge (\neg P \vee Q) \equiv ?$$

$$R?$$
$$\neg Q \vee Q \vee R?$$

Caution: you can only eliminate a single complementary variable in each time!

Conjunctive Normal Form (CNF)

- CNFs (conjunctions of disjunctions) play important roles in logical inference.

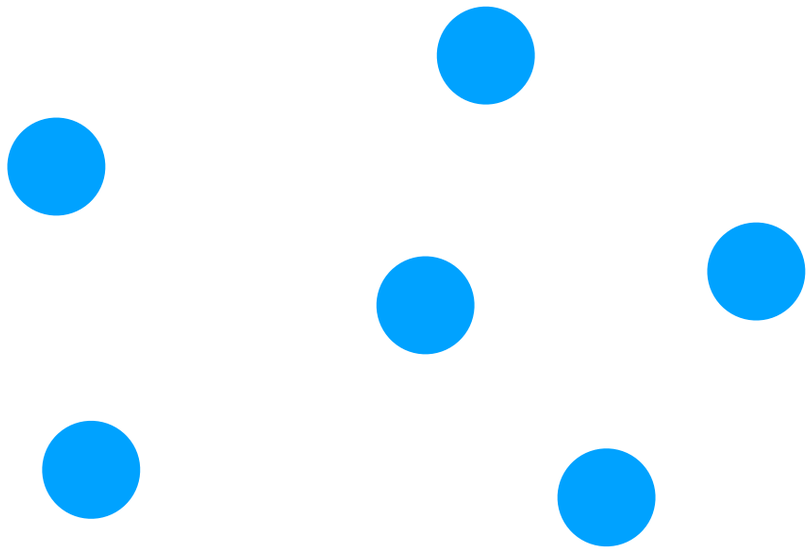
- Every sentence in propositional logic can be transformed to CNF

$$
\begin{aligned}
& (a \wedge b) \vee (c \wedge d) \\
\equiv & \neg(\neg a \vee \neg b) \vee \neg(\neg c \vee \neg d) \\
\equiv & (\neg a \vee \neg b) \wedge (\neg c \vee \neg d)
\end{aligned}
$$

- Even if the transformed CNF may much more complex, the CNF formulation will be very useful in the resolution algorithm.

The Resolution Algorithm

- Key idea: keep choosing pairs of disjunctions to do single-step resolution.

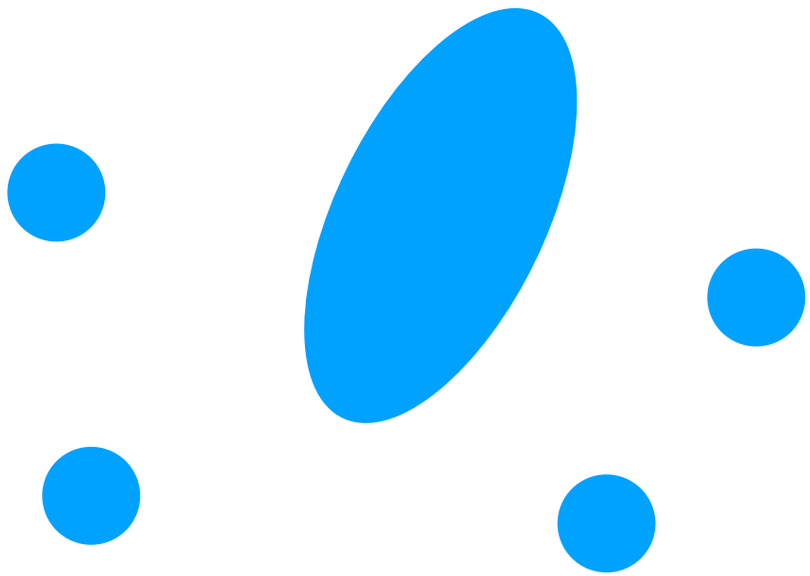


function PL-RESOLUTION(KB, α) **returns** *true* or *false*
  inputs: KB, the knowledge base, a sentence in propositional logic
    α, the query, a sentence in propositional logic

$clauses \leftarrow$ the set of clauses in the CNF representation of $KB \wedge \neg\alpha$
$new \leftarrow \{\}$
while *true* **do**
  for each pair of clauses C_i, C_j **in** *clauses* **do**
    $resolvents \leftarrow$ PL-RESOLVE(C_i, C_j)
    if *resolvents* contains the empty clause **then return** *true*
    $new \leftarrow new \cup resolvents$
  if $new \subseteq clauses$ **then return** *false*
  $clauses \leftarrow clauses \cup new$

The Resolution Algorithm

- Key idea: keep choosing pairs of disjunctions to do single-step resolution.

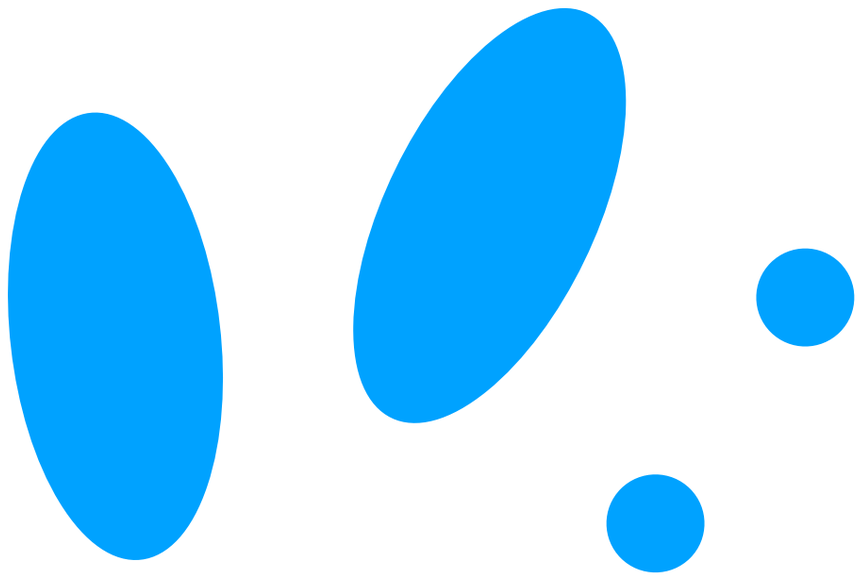


function PL-RESOLUTION(KB, α) **returns** *true* or *false*
  inputs: KB, the knowledge base, a sentence in propositional logic
    α, the query, a sentence in propositional logic

  clauses $\leftarrow$ the set of clauses in the CNF representation of $KB \wedge \neg\alpha$
  new $\leftarrow \{\}$
  while *true* **do**
    for each pair of clauses C_i, C_j **in** *clauses* **do**
      resolvents $\leftarrow$ PL-RESOLVE(C_i, C_j)
      if *resolvents* contains the empty clause **then return** *true*
      new $\leftarrow$ *new* $\cup$ *resolvents*
    if *new* $\subseteq$ *clauses* **then return** *false*
    clauses $\leftarrow$ *clauses* $\cup$ *new*

The Resolution Algorithm

- Key idea: keep choosing pairs of disjunctions to do single-step resolution.

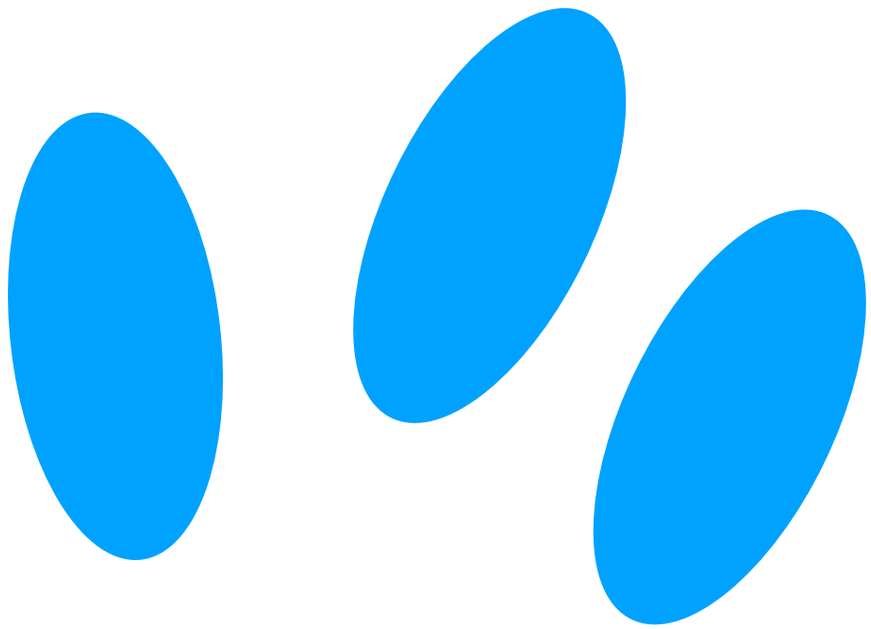


function PL-RESOLUTION(KB, α) **returns** *true* or *false*
  inputs: KB, the knowledge base, a sentence in propositional logic
    α, the query, a sentence in propositional logic

  $clauses \leftarrow$ the set of clauses in the CNF representation of $KB \wedge \neg\alpha$
  $new \leftarrow \{\}$
  while *true* **do**
    for each pair of clauses C_i, C_j **in** *clauses* **do**
      $resolvents \leftarrow$ PL-RESOLVE(C_i, C_j)
      if *resolvents* contains the empty clause **then return** *true*
      $new \leftarrow new \cup resolvents$
    if $new \subseteq clauses$ **then return** *false*
    $clauses \leftarrow clauses \cup new$

The Resolution Algorithm

- Key idea: keep choosing pairs of disjunctions to do single-step resolution.

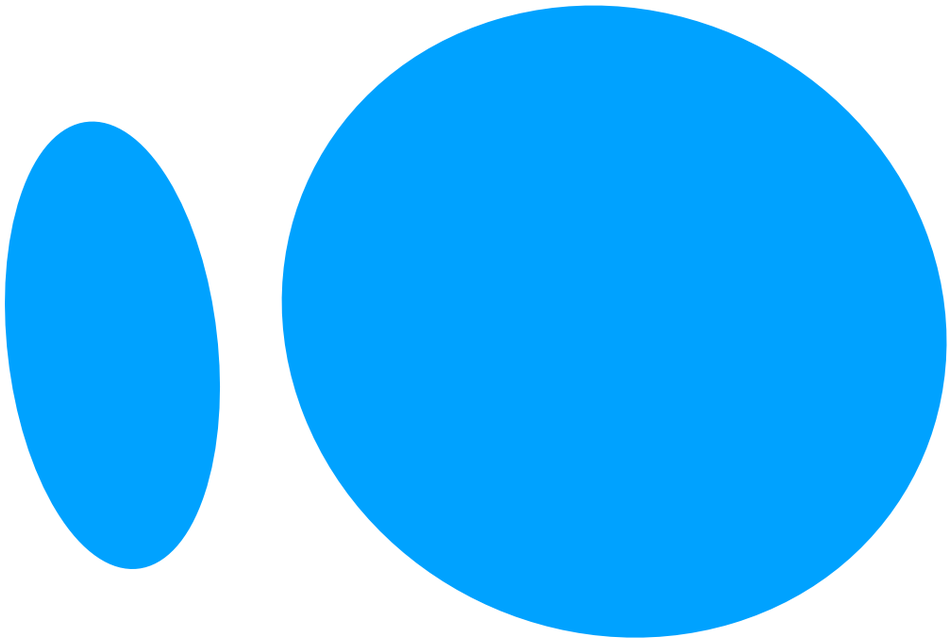


function PL-RESOLUTION(KB, α) **returns** *true* or *false*
  inputs: KB, the knowledge base, a sentence in propositional logic
    α, the query, a sentence in propositional logic

  $clauses \leftarrow$ the set of clauses in the CNF representation of $KB \wedge \neg\alpha$
  $new \leftarrow \{\}$
  while *true* **do**
    for each pair of clauses C_i, C_j **in** *clauses* **do**
      $resolvents \leftarrow$ PL-RESOLVE(C_i, C_j)
      if *resolvents* contains the empty clause **then return** *true*
      $new \leftarrow new \cup resolvents$
    if $new \subseteq clauses$ **then return** *false*
    $clauses \leftarrow clauses \cup new$

The Resolution Algorithm

- Key idea: keep choosing pairs of disjunctions to do single-step resolution.

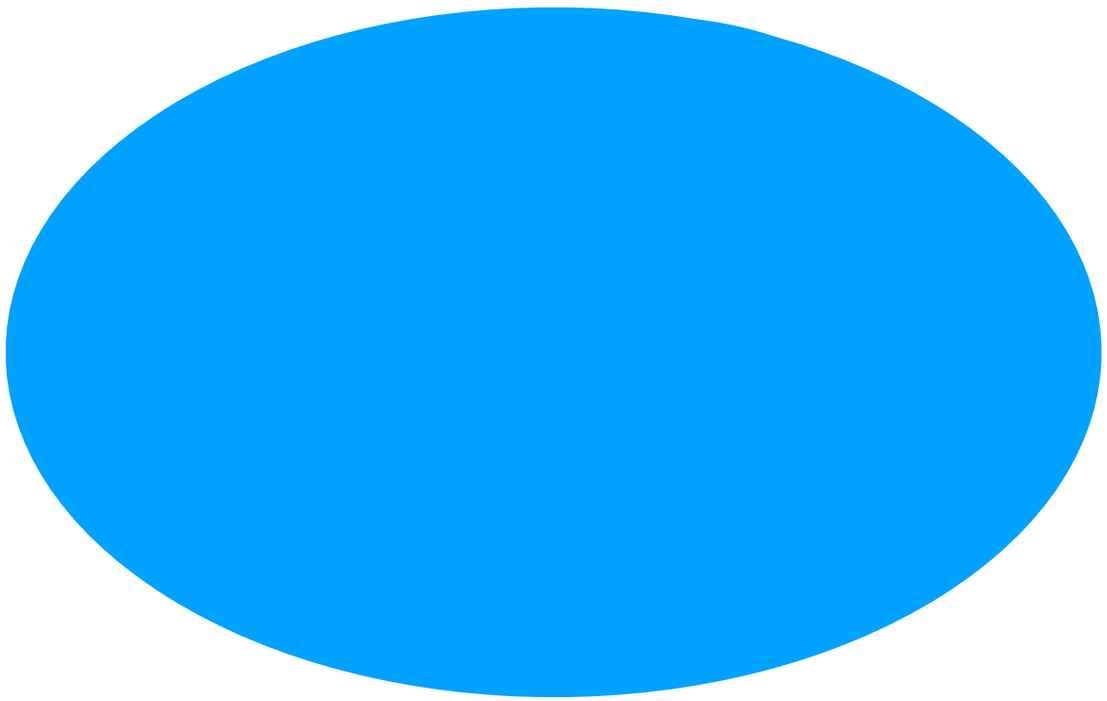


function PL-RESOLUTION(KB, α) **returns** *true* or *false*
  inputs: KB, the knowledge base, a sentence in propositional logic
    α, the query, a sentence in propositional logic

  $clauses \leftarrow$ the set of clauses in the CNF representation of $KB \wedge \neg\alpha$
  $new \leftarrow \{\}$
  while *true* **do**
    for each pair of clauses C_i, C_j **in** *clauses* **do**
      $resolvents \leftarrow$ PL-RESOLVE(C_i, C_j)
      if *resolvents* contains the empty clause **then return** *true*
      $new \leftarrow new \cup resolvents$
    if $new \subseteq clauses$ **then return** *false*
    $clauses \leftarrow clauses \cup new$

The Resolution Algorithm

- Key idea: keep choosing pairs of disjunctions to do single-step resolution.

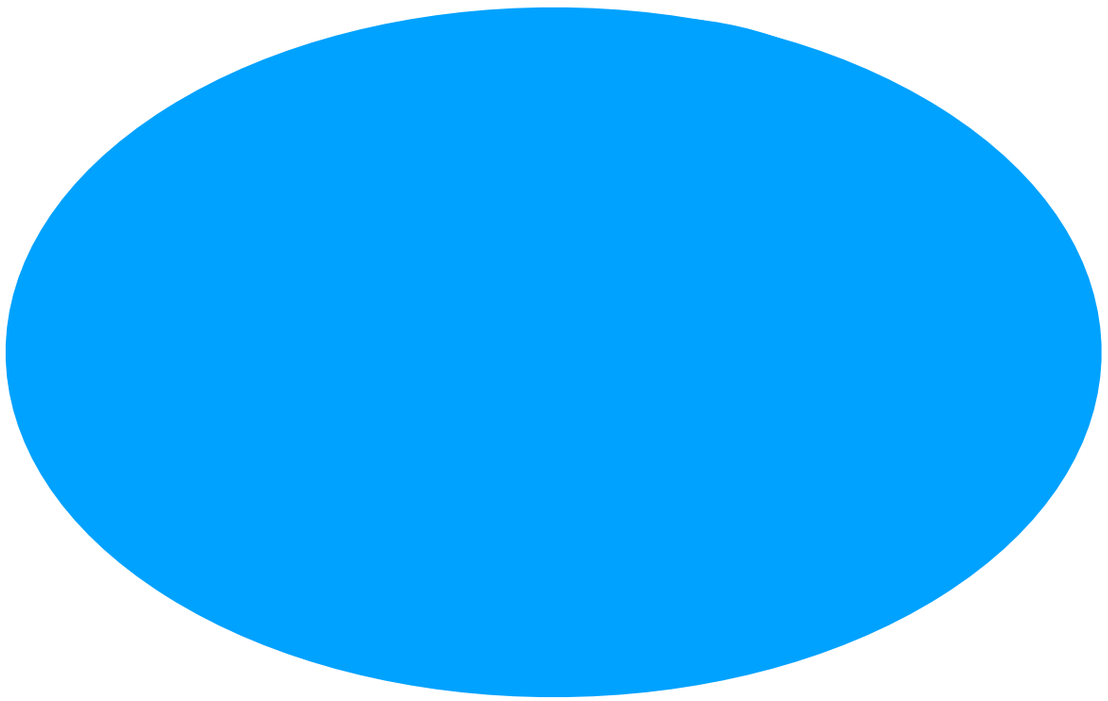


function PL-RESOLUTION(KB, α) **returns** *true* or *false*
  inputs: KB, the knowledge base, a sentence in propositional logic
    α, the query, a sentence in propositional logic

  $clauses \leftarrow$ the set of clauses in the CNF representation of $KB \wedge \neg\alpha$
  $new \leftarrow \{\}$
  while *true* **do**
    for each pair of clauses C_i, C_j **in** *clauses* **do**
      $resolvents \leftarrow$ PL-RESOLVE(C_i, C_j)
      if *resolvents* contains the empty clause **then return** *true*
      $new \leftarrow new \cup resolvents$
    if $new \subseteq clauses$ **then return** *false*
    $clauses \leftarrow clauses \cup new$

The Resolution Algorithm

- Key idea: keep choosing pairs of disjunctions to do single-step resolution.

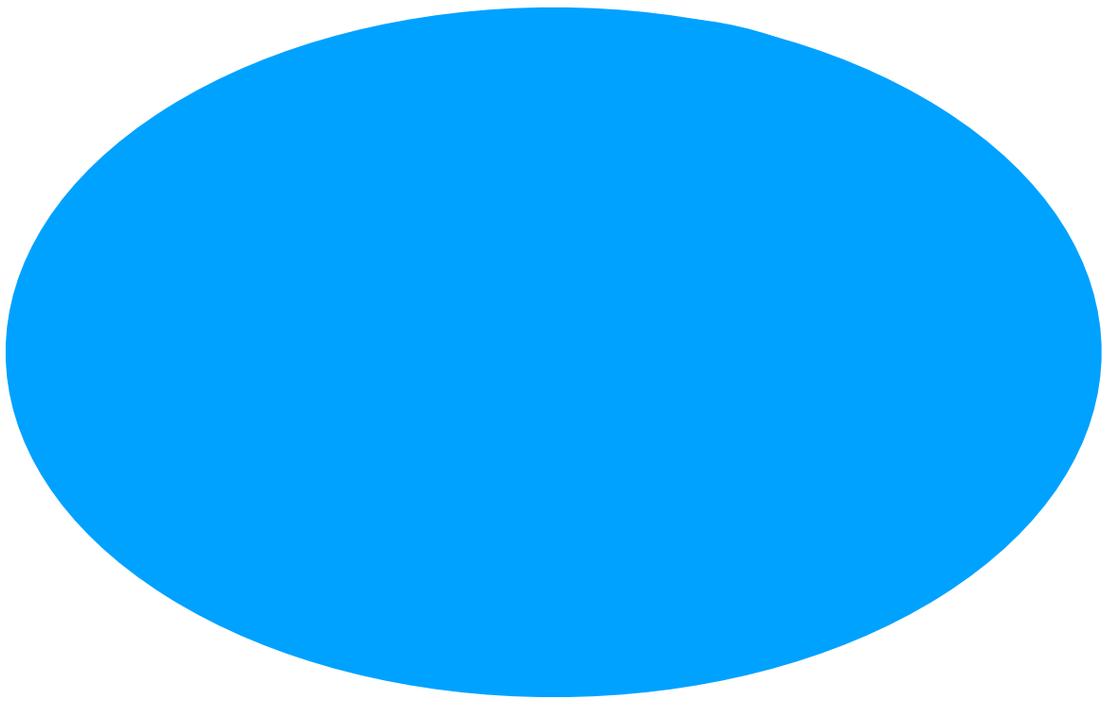


function PL-RESOLUTION(KB, α) **returns** *true* or *false*
  inputs: KB, the knowledge base, a sentence in propositional logic
    α, the query, a sentence in propositional logic

  $clauses \leftarrow$ the set of clauses in the CNF representation of $KB \wedge \neg\alpha$
  $new \leftarrow \{\}$
  while *true* **do**
    for each pair of clauses C_i, C_j **in** *clauses* **do**
      $resolvents \leftarrow$ PL-RESOLVE(C_i, C_j)
      if *resolvents* contains the empty clause **then return** *true*
      $new \leftarrow new \cup resolvents$
    if $new \subseteq clauses$ **then return** *false*
    $clauses \leftarrow clauses \cup new$

The Resolution Algorithm

- Key idea: keep choosing pairs of disjunctions to do single-step resolution.

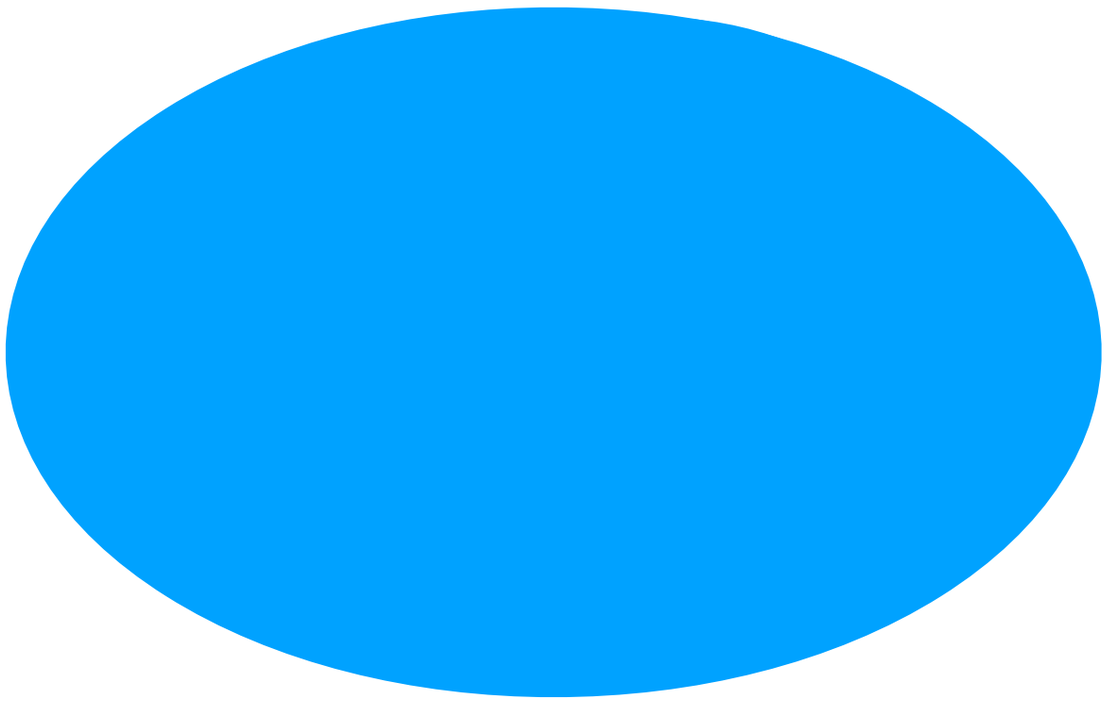


function PL-RESOLUTION(KB, α) **returns** *true* or *false*
  inputs: KB, the knowledge base, a sentence in propositional logic
    α, the query, a sentence in propositional logic

  $clauses \leftarrow$ the set of clauses in the CNF representation of $KB \wedge \neg\alpha$
  $new \leftarrow \{\}$
  while *true* **do**
    for each pair of clauses C_i, C_j **in** *clauses* **do**
      $resolvents \leftarrow$ PL-RESOLVE(C_i, C_j)
      if *resolvents* contains the empty clause **then return** *true*
      $new \leftarrow new \cup resolvents$
    if $new \subseteq clauses$ **then return** *false*
    $clauses \leftarrow clauses \cup new$

The Resolution Algorithm

- Key idea: keep choosing pairs of disjunctions to do single-step resolution.

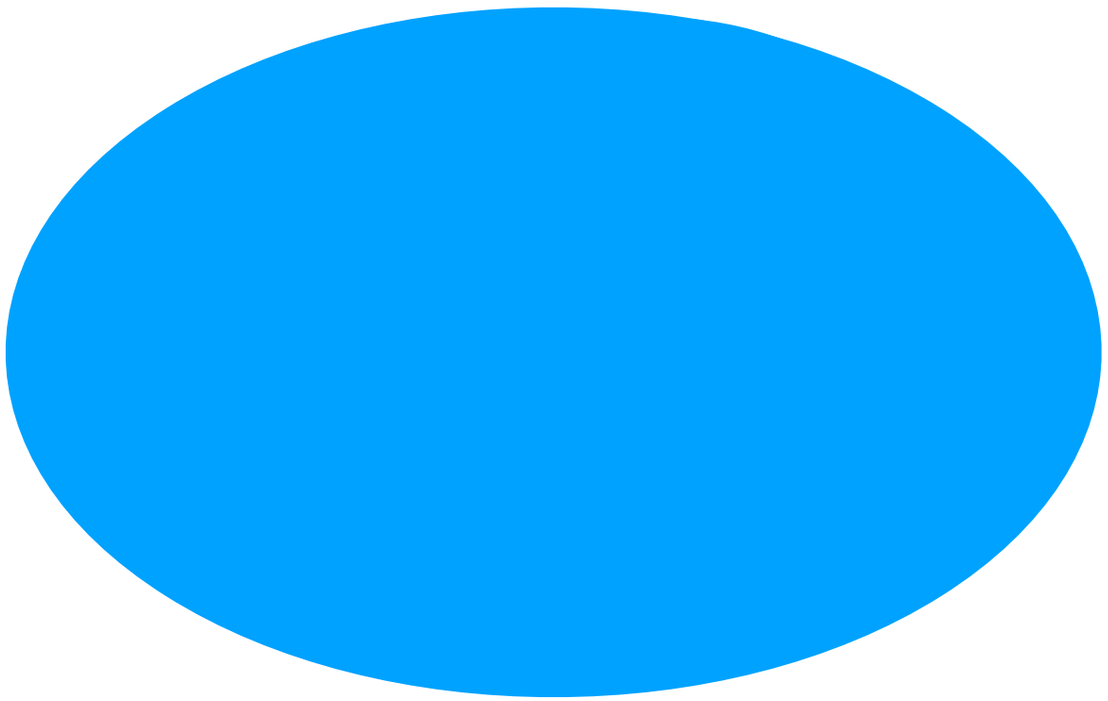


function PL-RESOLUTION(KB, α) **returns** *true* or *false*
  inputs: KB, the knowledge base, a sentence in propositional logic
    α, the query, a sentence in propositional logic

  clauses $\leftarrow$ the set of clauses in the CNF representation of $KB \wedge \neg\alpha$
  new $\leftarrow \{\}$
  while *true* **do**
    for each pair of clauses C_i, C_j **in** *clauses* **do**
      resolvents $\leftarrow$ PL-RESOLVE(C_i, C_j)
      if *resolvents* contains the empty clause **then return** *true*
      new $\leftarrow$ *new* $\cup$ *resolvents*
    if *new* $\subseteq$ *clauses* **then return** *false*
    clauses $\leftarrow$ *clauses* $\cup$ *new*

The Resolution Algorithm

- Key idea: keep choosing pairs of disjunctions to do single-step resolution.



function PL-RESOLUTION(KB, α) **returns** *true* or *false*
  inputs: KB, the knowledge base, a sentence in propositional logic
    α, the query, a sentence in propositional logic

  $clauses \leftarrow$ the set of clauses in the CNF representation of $KB \wedge \neg\alpha$
  $new \leftarrow \{\}$
  while *true* **do**
    for each pair of clauses C_i, C_j **in** *clauses* **do**
      $resolvents \leftarrow$ PL-RESOLVE(C_i, C_j)
      if *resolvents* contains the empty clause **then return** *true*
      $new \leftarrow new \cup resolvents$
    if $new \subseteq clauses$ **then return** *false*
    $clauses \leftarrow clauses \cup new$

Still exponential time in the worst case!

Completeness of Resolution

The ground resolution theorem:
If the CNF is unsatisfiable, then the resolution algorithm will output the empty clause (disjunction).

- Proof idea:
  - Prove the contrapositive: if the resolution algorithm does not output the empty clause, then the CNF is satisfiable.
  - Construct a model: for $i \leftarrow 1\ to\ k$, if in the solution, $\neg X_i$ is in some disjunction that all other literals are negative, then set it as false. Otherwise set it as true.
  - Use proof by contradiction to show that this model is satisfiable for the CNF (the error condition must be eliminated by the resolution algorithm)!

Knowledge Reasoning: I

- Logic Reasoning
- Propositional Logic
  - Search-Based Inference
  - Resolution-Based Inference
  - Forward and Backward Chaining
- First-Order Logic
- Take-Home Messages

Definite Clauses

- The resolution algorithm is powerful and general, but does not make use of the specific problem structures.

- In many problems, the logical sentences in both the knowledge database and the proof target are special ones:
  - Horn clause: disjunction with at most one literal is positive.
  - Definite clause: disjunction with exactly one literal is positive.

$$\neg X_1 \vee \neg X_2 \vee X_3$$

Definite Clauses

- The resolution algorithm is powerful and general, but does not make use of the specific problem structures.

- In many problems, the logical sentences in both the knowledge database and the proof target are special ones:
  - Horn clause: disjunction with at most one literal is positive.
  - Definite clause: disjunction with exactly one literal is positive.

$$\neg X_1 \vee \neg X_2 \vee X_3$$

Why this kind of clauses are interesting?

Definite Clauses

- The resolution algorithm is powerful and general, but does not make use of the specific problem structures.

- In many problems, the logical sentences in both the knowledge database and the proof target are special ones:
  - Horn clause: disjunction with at most one literal is positive.
  - Definite clause: disjunction with exactly one literal is positive.

$$\neg X_1 \vee \neg X_2 \vee X_3$$

Why this kind of clauses are interesting?

$$\neg X_1 \vee \neg X_2 \vee X_3 \equiv (X_1 \wedge X_2) \Rightarrow X_3$$

Definite Clauses

- The resolution algorithm is powerful and general, but does not make use of the specific problem structures.

- In many problems, the logical sentences in both the knowledge database and the proof target are special ones:
  - Horn clause: disjunction with at most one literal is positive.
  - Definite clause: disjunction with exactly one literal is positive.

$$\neg X_1 \vee \neg X_2 \vee X_3$$

Why this kind of clauses are interesting?

$$\neg X_1 \vee \neg X_2 \vee X_3 \equiv (X_1 \wedge X_2) \Rightarrow X_3$$

Commonly seen in real-world problems!

Forward Chaining Algorithm

Want to prove Q (definite clauses)

CLAUSES	*COUNT*
▪ P ⇒ Q	1
▪ L ∧ M ⇒ P	2
▪ B ∧ L ⇒ M	2
▪ A ∧ P ⇒ L	2
▪ A ∧ B ⇒ L	2
▪ A	0
▪ B	0

QUEUE

INFERRED

A

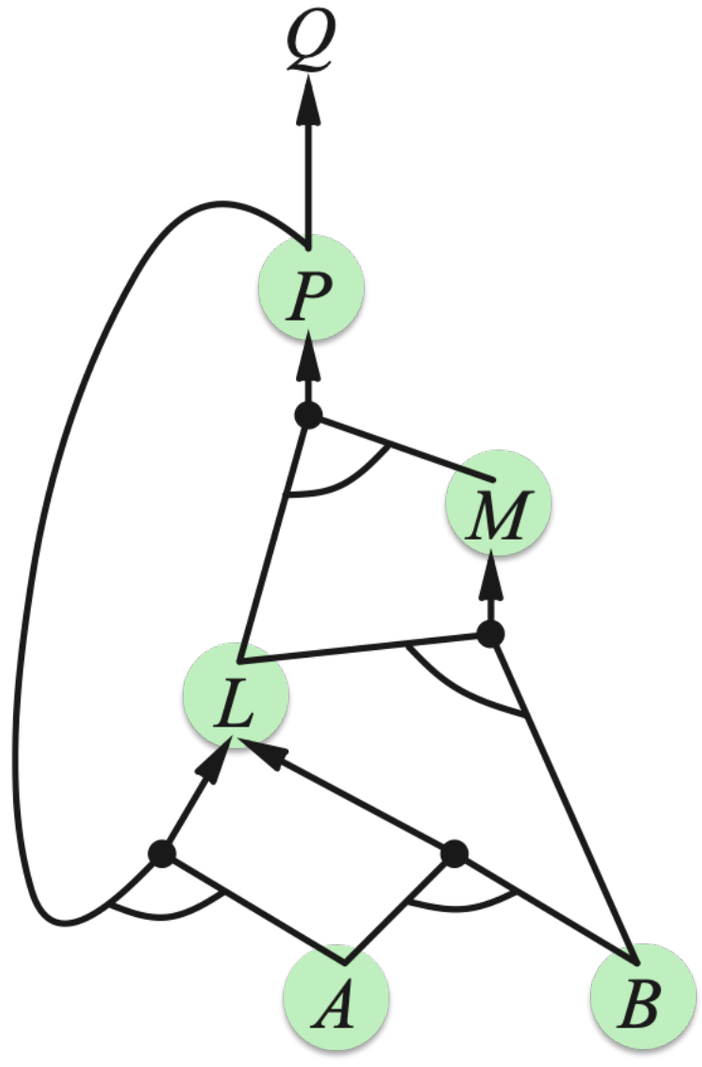
B

L

M

P

Q



Forward Chaining Algorithm

Want to prove Q (definite clauses)

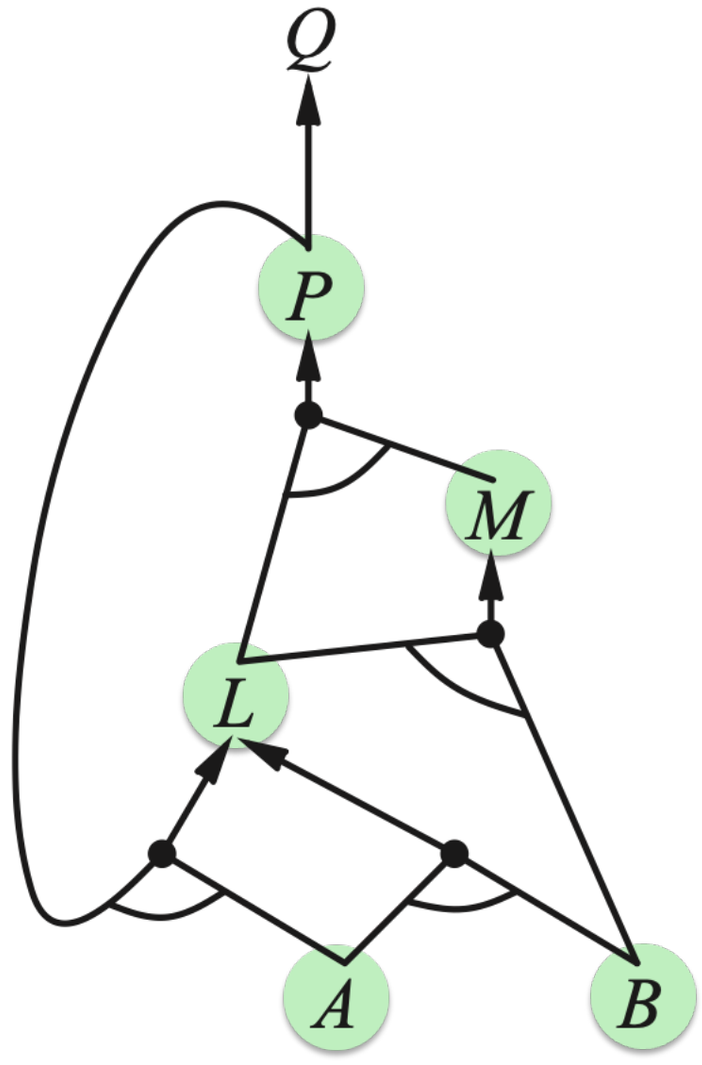
CLAUSES	*COUNT*
▪ P ⇒ Q	1
▪ L ∧ M ⇒ P	2
▪ B ∧ L ⇒ M	2
▪ A ∧ P ⇒ L	2
▪ A ∧ B ⇒ L	2
▪ A	0
▪ B	0

INFERRED

A
B
L
M
P
Q

QUEUE

A B



Forward Chaining Algorithm

Want to prove Q (definite clauses)

CLAUSES	COUNT
▪ P ⇒ Q	1
▪ L ∧ M ⇒ P	2
▪ B ∧ L ⇒ M	2
▪ A ∧ P ⇒ L	2
▪ A ∧ B ⇒ L	2
▪ A	0
▪ B	0

INFERRED

A ✓

B ✓

L

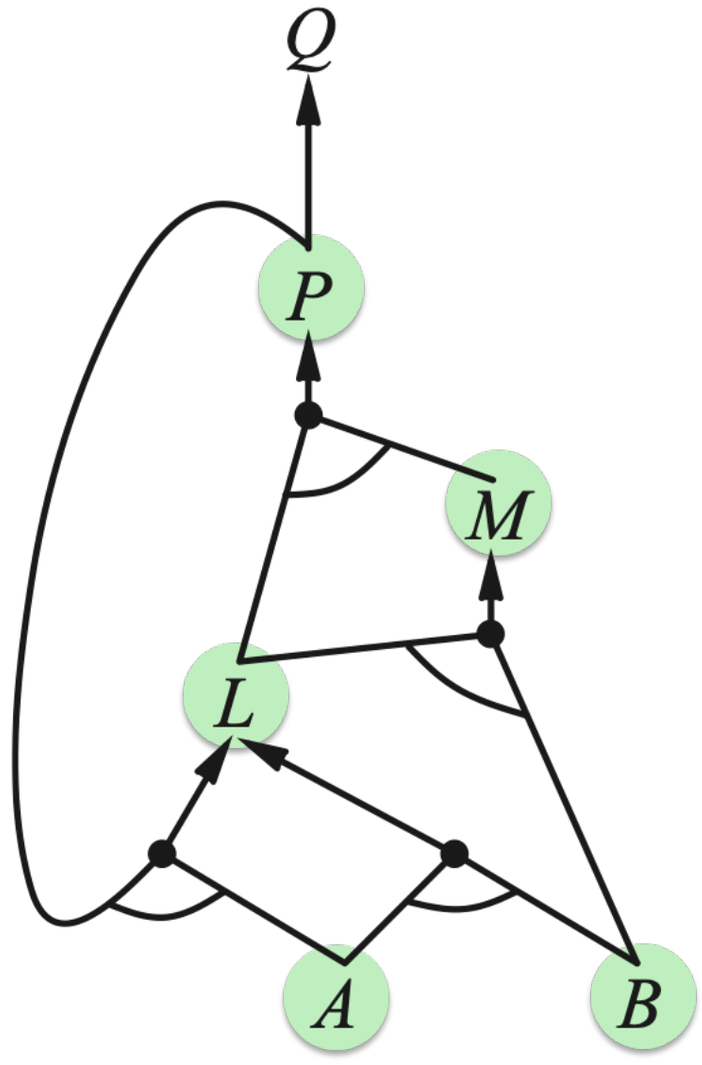
M

P

Q

QUEUE

A B



Forward Chaining Algorithm

Want to prove Q (definite clauses)

CLAUSES	COUNT
▪ P ⇒ Q	1
▪ L ∧ M ⇒ P	2
▪ B ∧ L ⇒ M	1
▪ A ∧ P ⇒ L	1
▪ A ∧ B ⇒ L	0
▪ A	0
▪ B	0

INFERRED

A ✓

B ✓

L

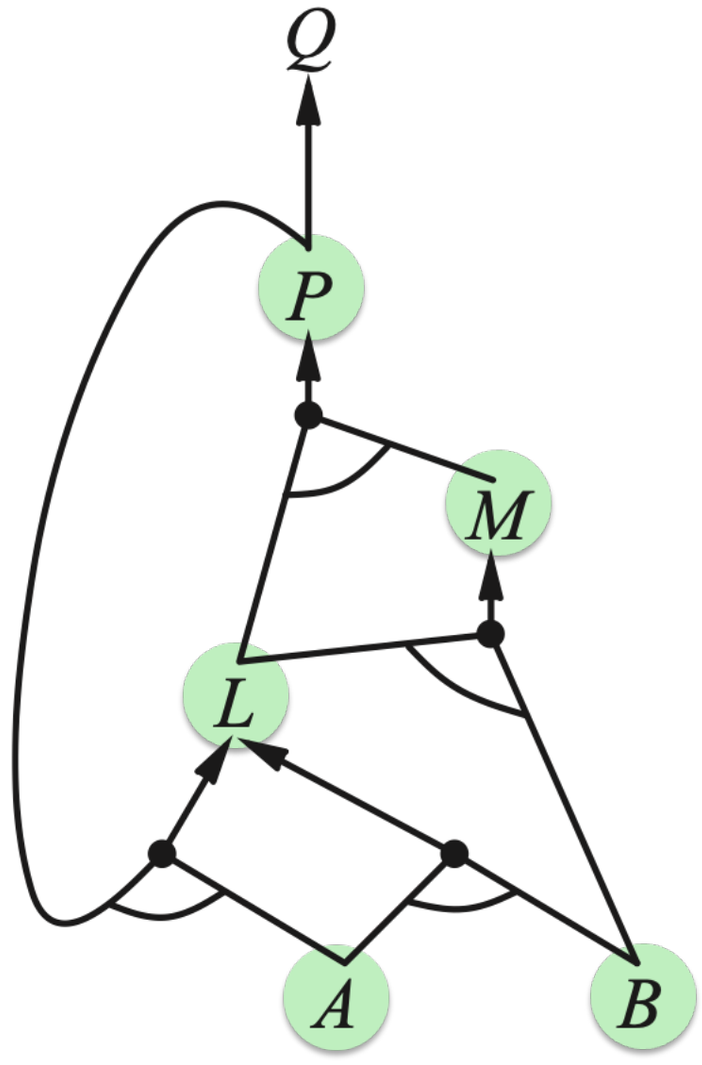
M

P

Q

QUEUE

A B



Forward Chaining Algorithm

Want to prove Q (definite clauses)

CLAUSES	*COUNT*
▪ P ⇒ Q	1
▪ L ∧ M ⇒ P	2
▪ B ∧ L ⇒ M	1
▪ A ∧ P ⇒ L	1
▪ A ∧ B ⇒ L	0
▪ A	0
▪ B	0

INFERRED

A ✓

B ✓

L

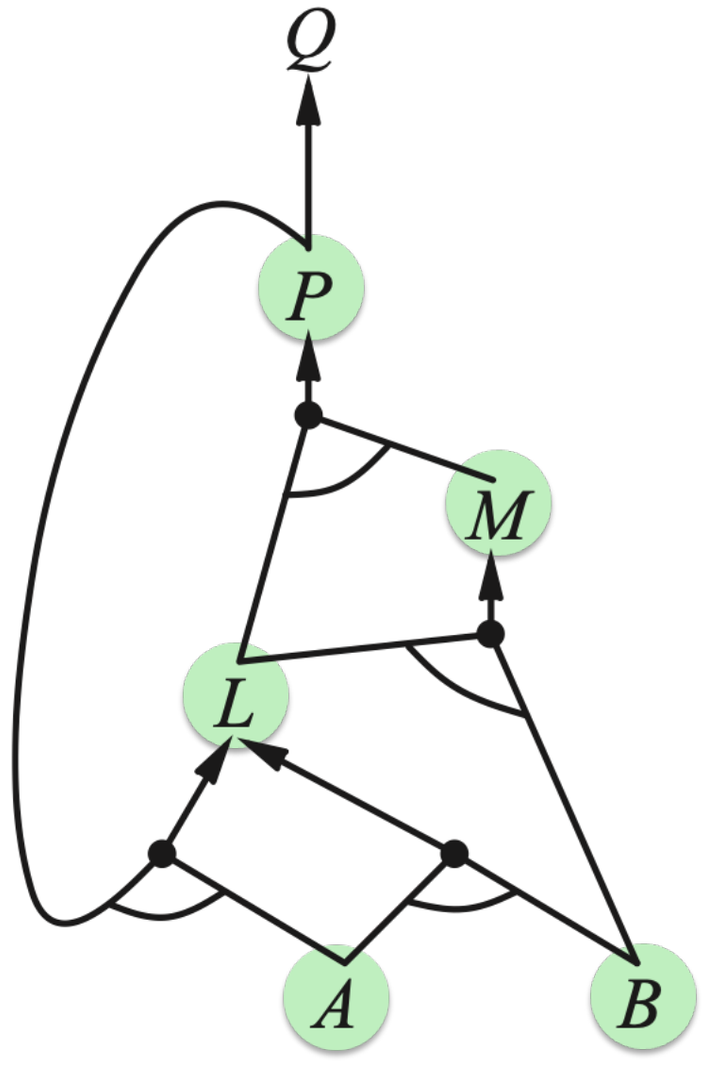
M

P

Q

QUEUE

A B L



Forward Chaining Algorithm

Want to prove Q (definite clauses)

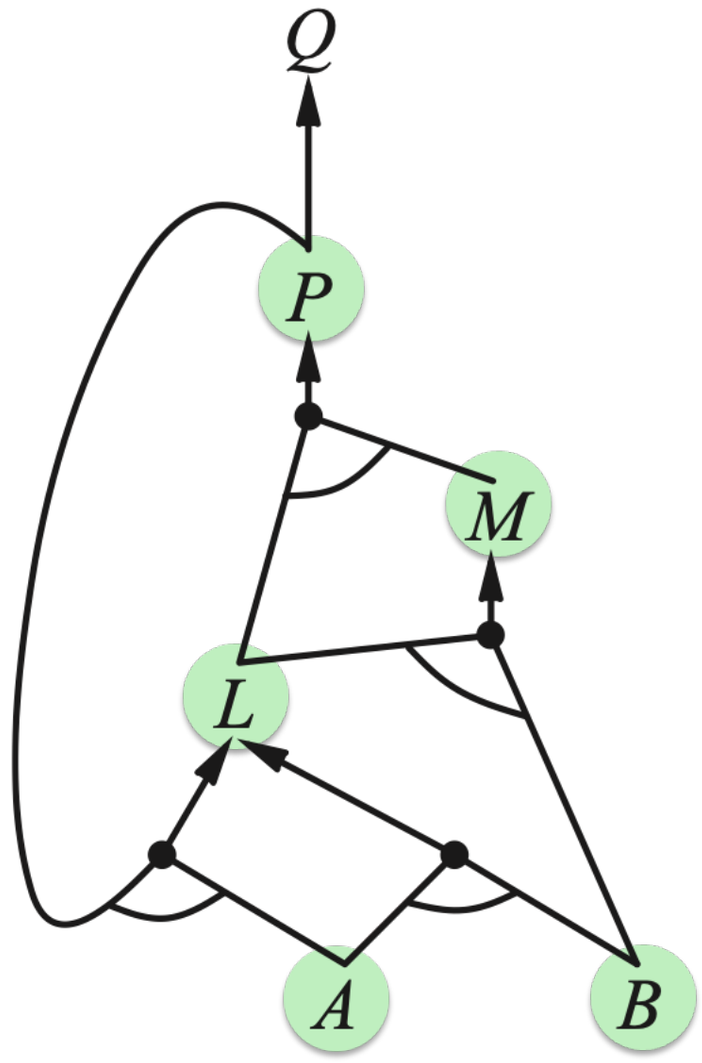
CLAUSES	COUNT
▪ P ⇒ Q	1
▪ L ∧ M ⇒ P	2
▪ B ∧ L ⇒ M	1
▪ A ∧ P ⇒ L	1
▪ A ∧ B ⇒ L	0
▪ A	0
▪ B	0

INFERRED

A ✓
B ✓
L ✓
M
P
Q

QUEUE

A B L



Forward Chaining Algorithm

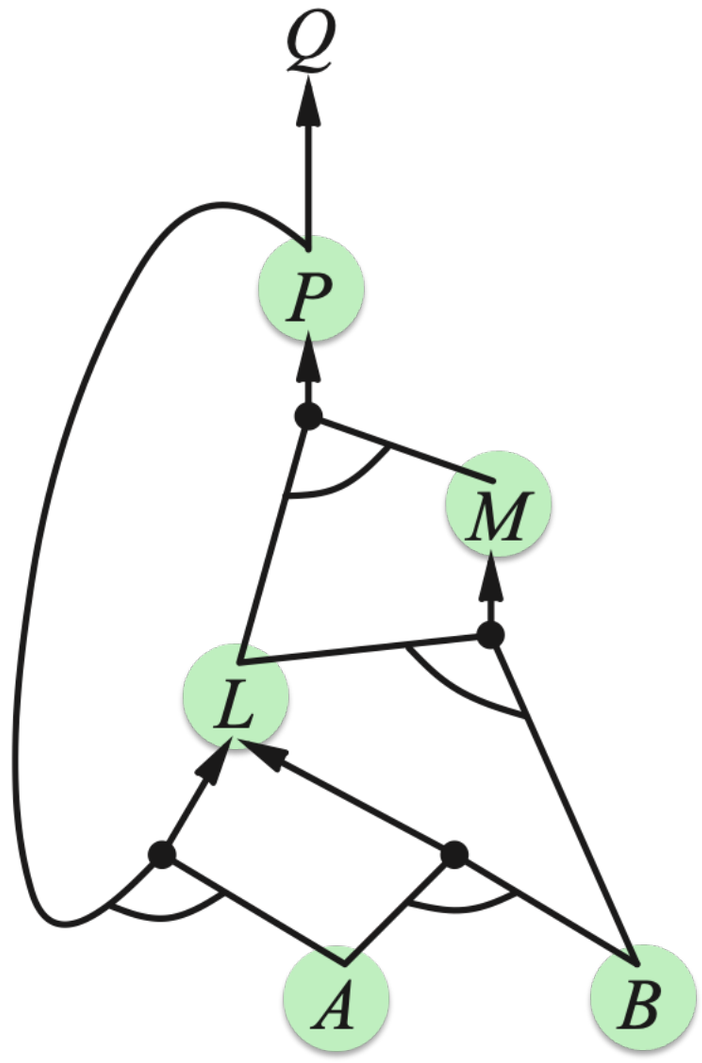
Want to prove Q (definite clauses)

CLAUSES	COUNT
▪ P ⇒ Q	1
▪ L ∧ M ⇒ P	1
▪ B ∧ L ⇒ M	0
▪ A ∧ P ⇒ L	1
▪ A ∧ B ⇒ L	0
▪ A	0
▪ B	0

INFERRED
A ✓
B ✓
L ✓
M
P
Q

QUEUE

A B L



Forward Chaining Algorithm

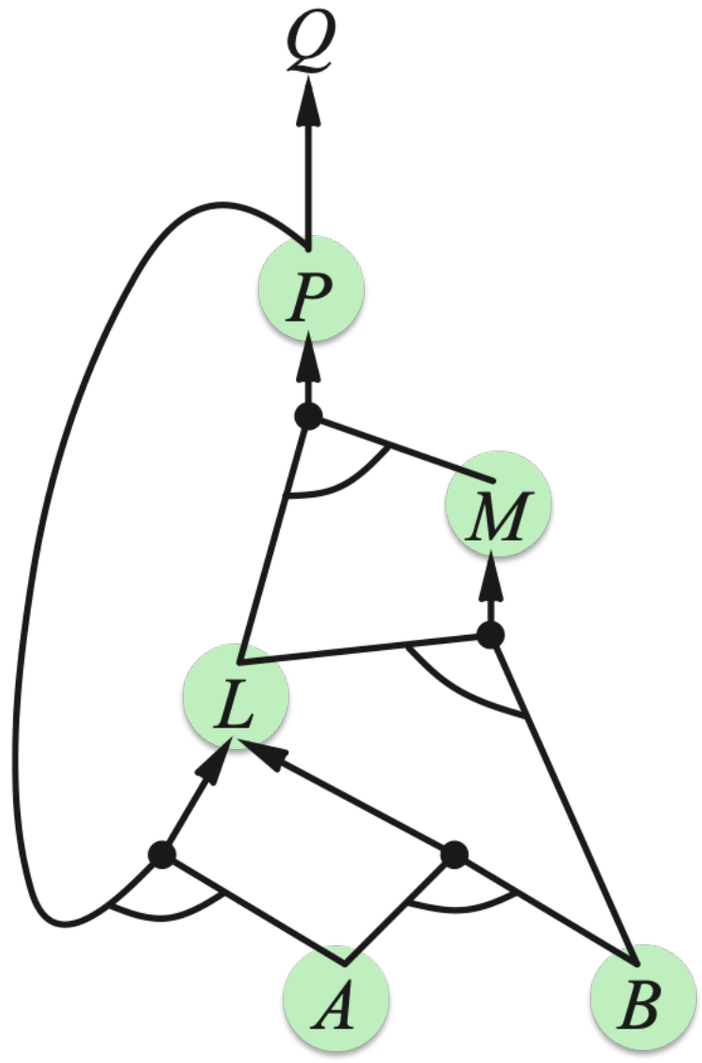
Want to prove Q (definite clauses)

CLAUSES	COUNT
▪ P ⇒ Q	1
▪ L ∧ M ⇒ P	1
▪ B ∧ L ⇒ M	0
▪ A ∧ P ⇒ L	1
▪ A ∧ B ⇒ L	0
▪ A	0
▪ B	0

INFERRED
A ✓
B ✓
L ✓
M
P
Q

QUEUE

A B L M



Forward Chaining Algorithm

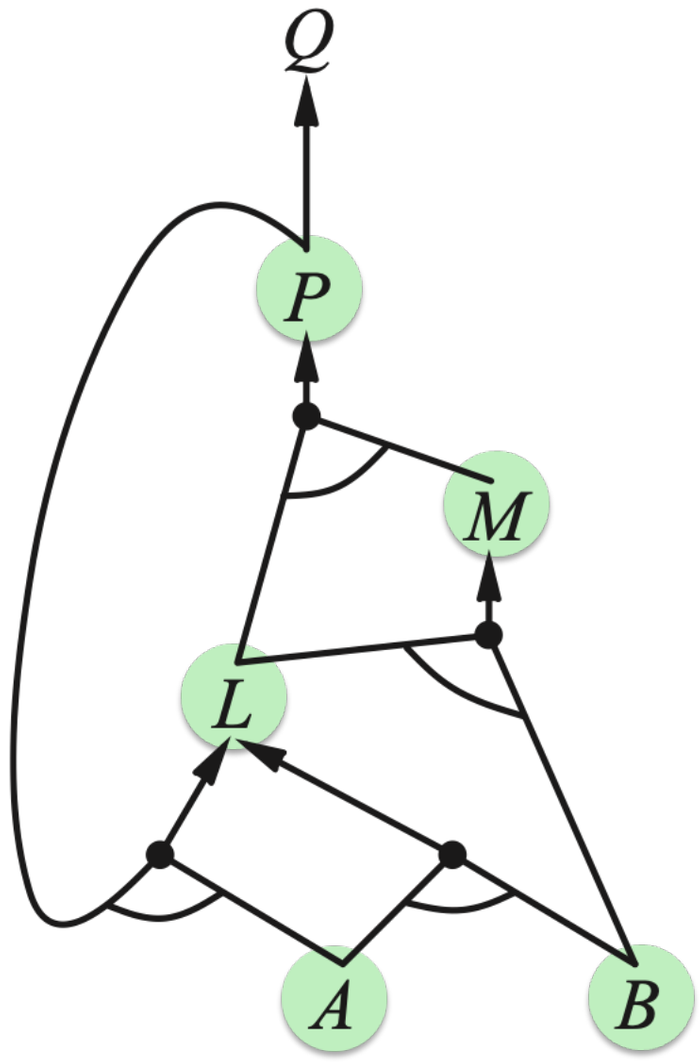
Want to prove Q (definite clauses)

CLAUSES	COUNT
▪ P ⇒ Q	1
▪ L ∧ M ⇒ P	1
▪ B ∧ L ⇒ M	0
▪ A ∧ P ⇒ L	1
▪ A ∧ B ⇒ L	0
▪ A	0
▪ B	0

INFERRED
A ✓
B ✓
L ✓
M ✓
P
Q

QUEUE

A B L M



Forward Chaining Algorithm

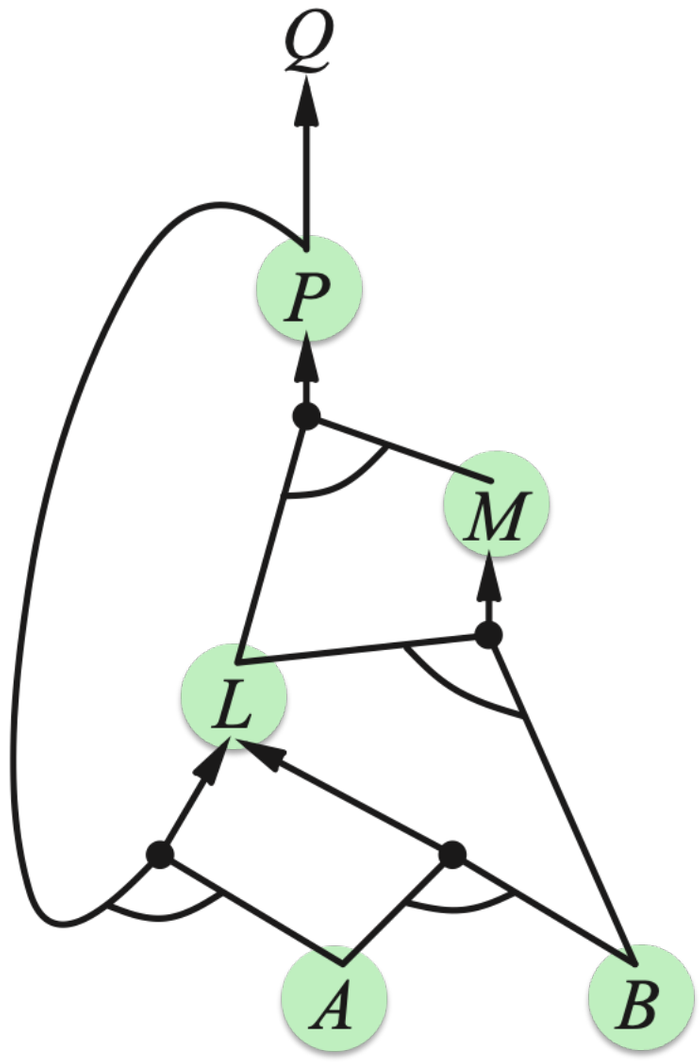
Want to prove Q (definite clauses)

CLAUSES	COUNT
▪ P ⇒ Q	1
▪ L ∧ M ⇒ P	0
▪ B ∧ L ⇒ M	0
▪ A ∧ P ⇒ L	1
▪ A ∧ B ⇒ L	0
▪ A	0
▪ B	0

INFERRED
A ✓
B ✓
L ✓
M ✓
P
Q

QUEUE

A B L M



Forward Chaining Algorithm

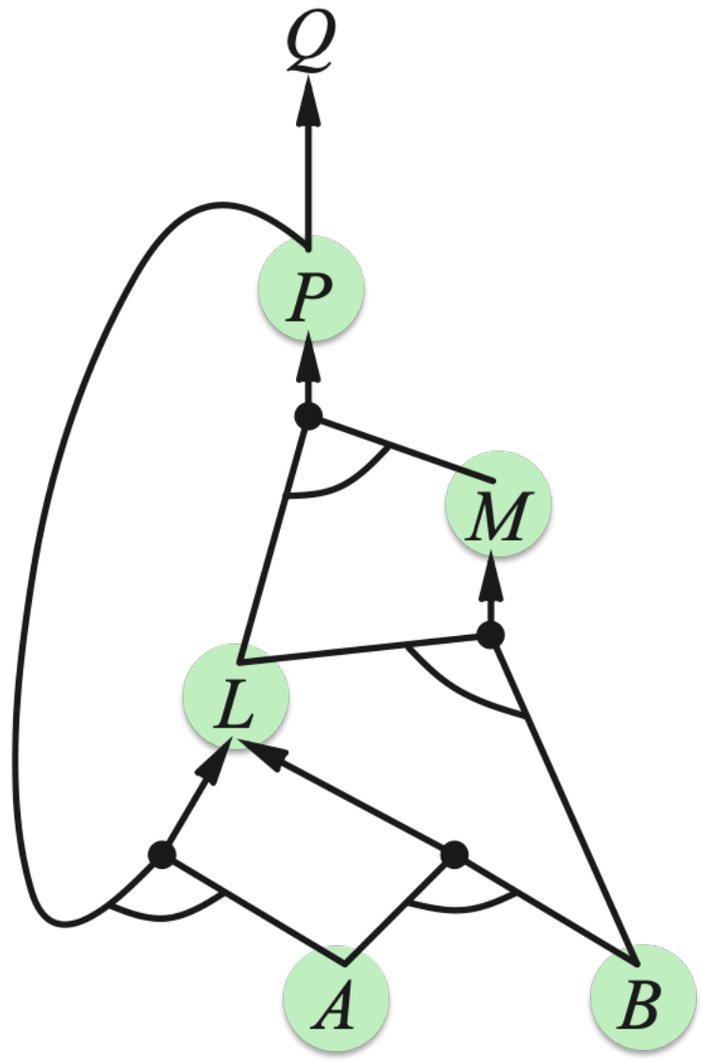
Want to prove Q (definite clauses)

CLAUSES	COUNT
▪ P ⇒ Q	1
▪ L ∧ M ⇒ P	0
▪ B ∧ L ⇒ M	0
▪ A ∧ P ⇒ L	1
▪ A ∧ B ⇒ L	0
▪ A	0
▪ B	0

INFERRED
A ✓
B ✓
L ✓
M ✓
P
Q

QUEUE

A B L M P



Forward Chaining Algorithm

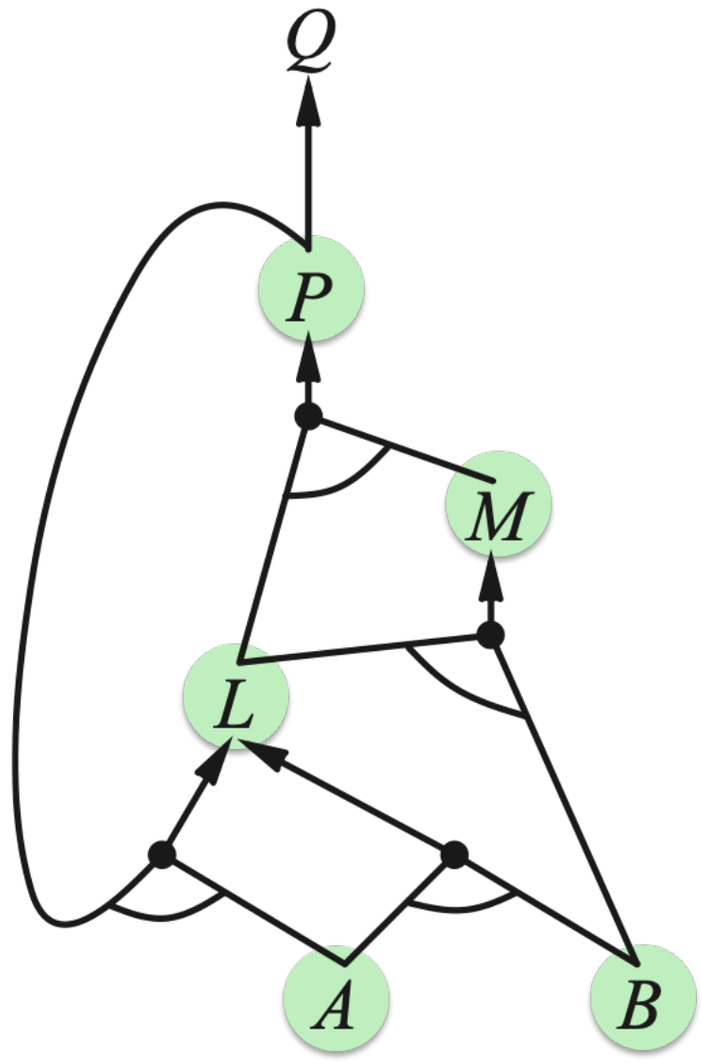
Want to prove Q (definite clauses)

CLAUSES	COUNT
▪ P ⇒ Q	1
▪ L ∧ M ⇒ P	0
▪ B ∧ L ⇒ M	0
▪ A ∧ P ⇒ L	1
▪ A ∧ B ⇒ L	0
▪ A	0
▪ B	0

INFERRED
A ✓
B ✓
L ✓
M ✓
P ✓
Q

QUEUE

A B L M P



Forward Chaining Algorithm

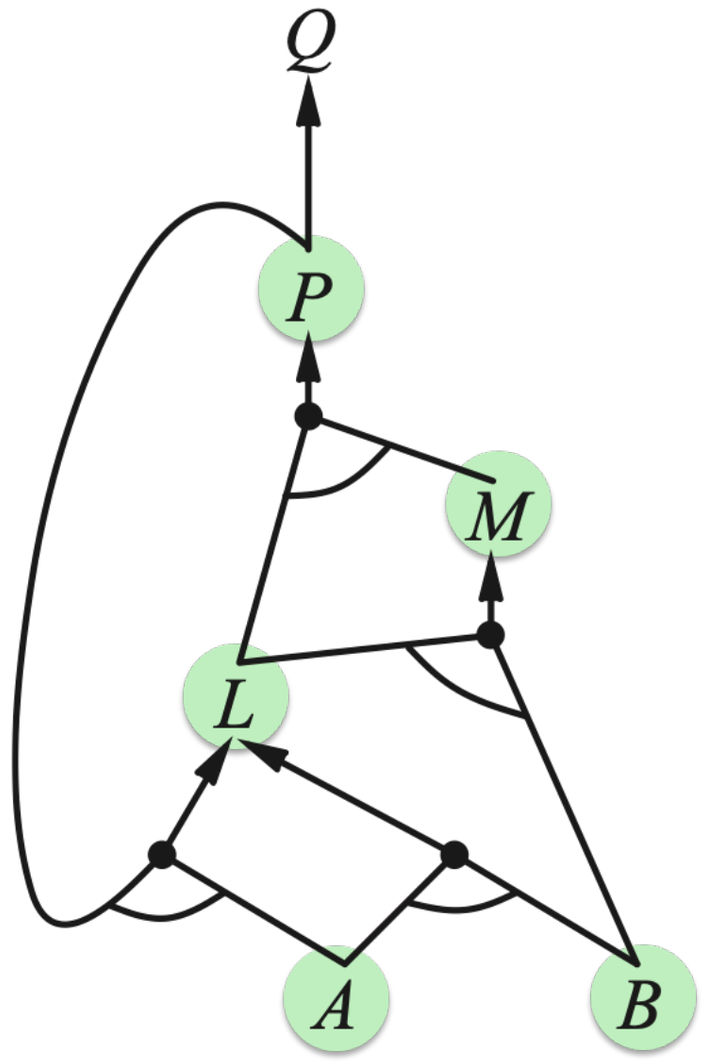
Want to prove Q (definite clauses)

CLAUSES	COUNT
▪ P ⇒ Q	0
▪ L ∧ M ⇒ P	0
▪ B ∧ L ⇒ M	0
▪ A ∧ P ⇒ L	0
▪ A ∧ B ⇒ L	0
▪ A	0
▪ B	0

INFERRED
A ✓
B ✓
L ✓
M ✓
P ✓
Q

QUEUE

A B L M P



Forward Chaining Algorithm

Want to prove Q (definite clauses)

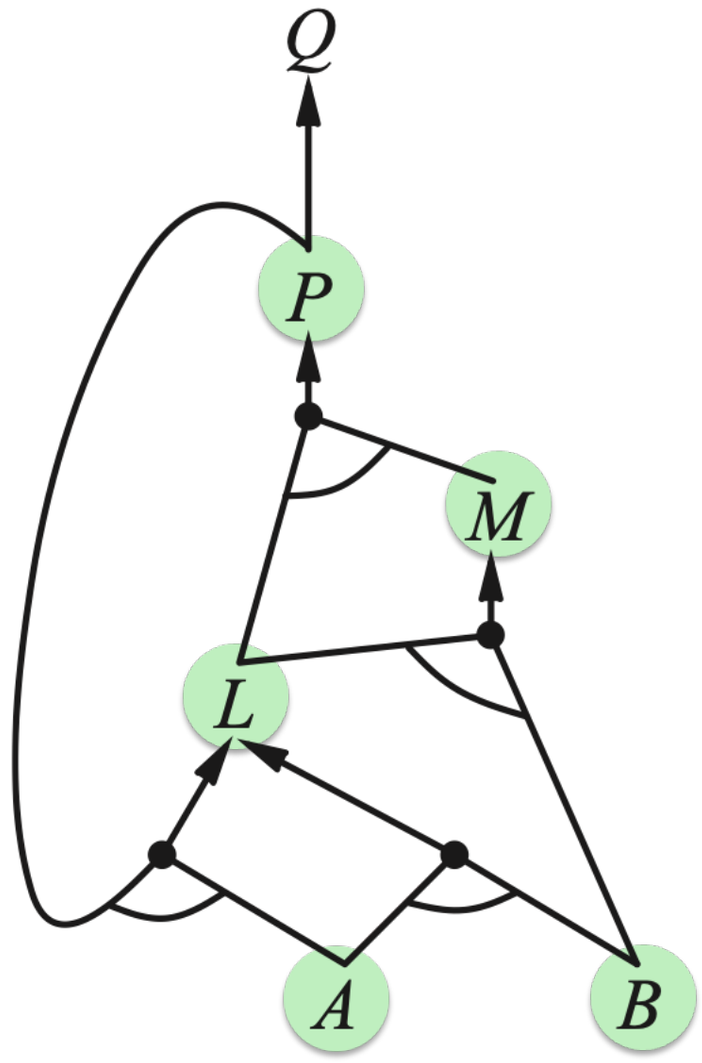
CLAUSES	COUNT
▪ P ⇒ Q	0
▪ L ∧ M ⇒ P	0
▪ B ∧ L ⇒ M	0
▪ A ∧ P ⇒ L	0
▪ A ∧ B ⇒ L	0
▪ A	0
▪ B	0

INFERRED

A ✓
B ✓
L ✓
M ✓
P ✓
Q

QUEUE

A B L M P Q



Forward Chaining Algorithm

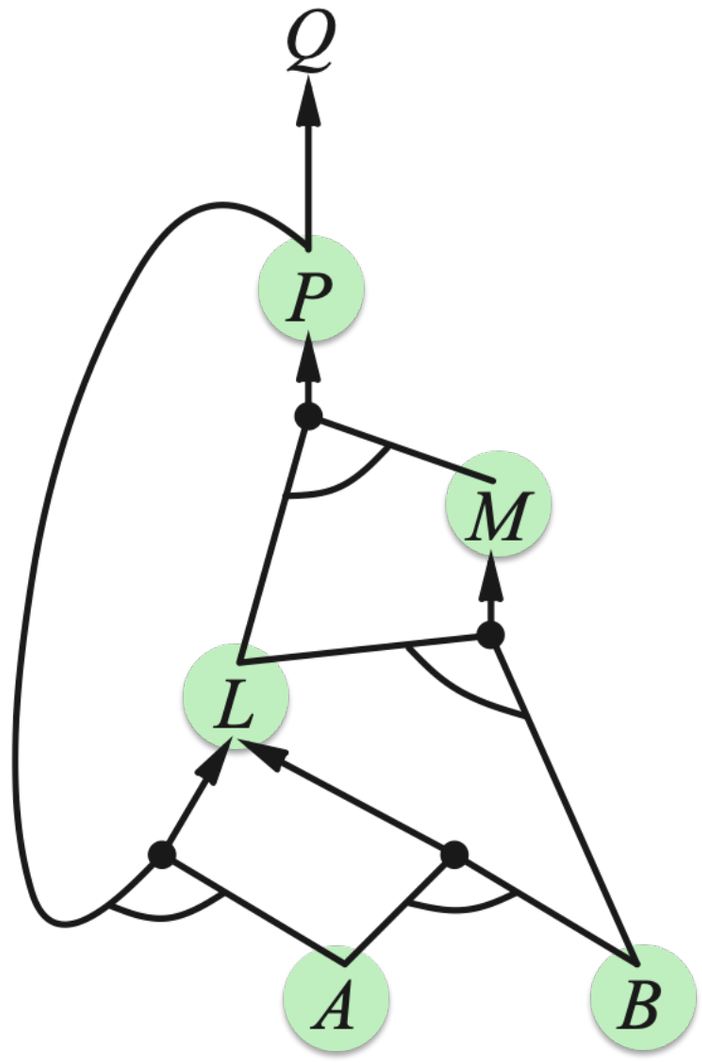
Want to prove Q (definite clauses)

CLAUSES	COUNT
▪ P ⇒ Q	0
▪ L ∧ M ⇒ P	0
▪ B ∧ L ⇒ M	0
▪ A ∧ P ⇒ L	0
▪ A ∧ B ⇒ L	0
▪ A	0
▪ B	0

INFERRED
A ✓
B ✓
L ✓
M ✓
P ✓
Q ✓

QUEUE

A B L M P Q



Forward Chaining Algorithm

function PL-FC-ENTAILS?(KB, q) **returns** *true* or *false*

inputs: KB, the knowledge base, a set of propositional definite clauses
q, the query, a proposition symbol

$count \leftarrow$ a table, where $count[c]$ is initially the number of symbols in clause c's premise
$inferred \leftarrow$ a table, where $inferred[s]$ is initially *false* for all symbols
$queue \leftarrow$ a queue of symbols, initially symbols known to be true in KB

while $queue$ is not empty **do**
  $p \leftarrow$ POP($queue$)
  if $p = q$ **then return** *true*
  if $inferred[p] = false$ **then**
    $inferred[p] \leftarrow true$
    for each clause c in KB where p is in c.PREMISE **do**
      decrement $count[c]$
      if $count[c] = 0$ **then** add c.CONCLUSION to $queue$
return *false*

Forward Chaining Algorithm

function PL-FC-ENTAILS?(KB, q) **returns** *true* or *false*

inputs: KB, the knowledge base, a set of propositional definite clauses
q, the query, a proposition symbol

$count \leftarrow$ a table, where $count[c]$ is initially the number of symbols in clause c's premise
$inferred \leftarrow$ a table, where $inferred[s]$ is initially *false* for all symbols
$queue \leftarrow$ a queue of symbols, initially symbols known to be true in KB

while $queue$ is not empty **do**
  $p \leftarrow$ POP($queue$)
  if $p = q$ **then return** *true*
  if $inferred[p] = false$ **then**
    $inferred[p] \leftarrow true$
    for each clause c in KB where p is in c.PREMISE **do**
      decrement $count[c]$
      if $count[c] = 0$ **then** add c.CONCLUSION to $queue$
return *false*

Forward Chaining Algorithm

function PL-FC-ENTAILS?(KB, q) **returns** *true* or *false*

inputs: KB, the knowledge base, a set of propositional definite clauses
q, the query, a proposition symbol

$count \leftarrow$ a table, where $count[c]$ is initially the number of symbols in clause c's premise
$inferred \leftarrow$ a table, where $inferred[s]$ is initially *false* for all symbols
$queue \leftarrow$ a queue of symbols, initially symbols known to be true in KB

while $queue$ is not empty **do**
  $p \leftarrow$ POP($queue$)
  if $p = q$ **then return** *true*
  if $inferred[p] = false$ **then**
    $inferred[p] \leftarrow true$
    for each clause c in KB where p is in c.PREMISE **do**
      decrement $count[c]$
      if $count[c] = 0$ **then** add c.CONCLUSION to $queue$
return *false*

Forward Chaining Algorithm

function PL-FC-ENTAILS?(KB, q) **returns** *true* or *false*

inputs: KB, the knowledge base, a set of propositional definite clauses
q, the query, a proposition symbol

$count \leftarrow$ a table, where $count[c]$ is initially the number of symbols in clause c's premise
$inferred \leftarrow$ a table, where $inferred[s]$ is initially *false* for all symbols
$queue \leftarrow$ a queue of symbols, initially symbols known to be true in KB

while $queue$ is not empty **do**
  $p \leftarrow$ POP($queue$)
  if $p = q$ **then return** *true*
  if $inferred[p] = false$ **then**
    $inferred[p] \leftarrow true$
    for each clause c in KB where p is in c.PREMISE **do**
      decrement $count[c]$
      if $count[c] = 0$ **then** add c.CONCLUSION to $queue$
return *false*

Forward Chaining Algorithm

function PL-FC-ENTAILS?(KB, q) **returns** *true* or *false*

inputs: KB, the knowledge base, a set of propositional definite clauses
q, the query, a proposition symbol

$count \leftarrow$ a table, where $count[c]$ is initially the number of symbols in clause c's premise
$inferred \leftarrow$ a table, where $inferred[s]$ is initially *false* for all symbols
$queue \leftarrow$ a queue of symbols, initially symbols known to be true in KB

while $queue$ is not empty **do**
  $p \leftarrow$ POP($queue$)
  if $p = q$ **then return** *true*
  if $inferred[p] = false$ **then**
    $inferred[p] \leftarrow true$
    for each clause c in KB where p is in c.PREMISE **do**
      decrement $count[c]$
      if $count[c] = 0$ **then** add c.CONCLUSION to $queue$
return *false*

Completeness of Forward Chaining

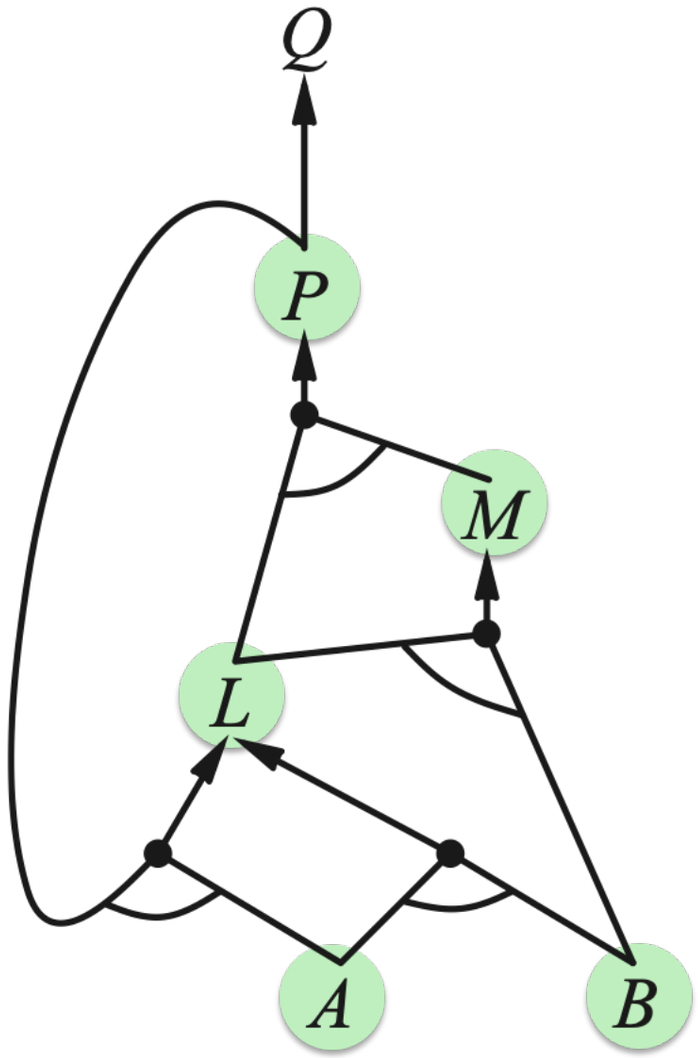
- Theorem: FC is sound and complete for definite-clause KBs
- Soundness: follows from soundness of Modus Ponens (easy to check)
- Completeness proof:

1. FC reaches a fixed point where no new atomic sentences are derived
2. Consider the final *inferred* table as a model ***m***, assigning true/false to symbols
3. Every clause in the original KB is true in ***m***

   Proof: Suppose a clause $a_1 \wedge \ldots \wedge a_k \Rightarrow b$ is false in ***m***
   Then $a_1 \wedge \ldots \wedge a_k$ is true in ***m*** and b is false in ***m***
   Therefore the algorithm has not reached a fixed point!
4. Hence ***m*** is a model of KB
5. If KB $\models q$, q is true in every model of KB, including ***m***

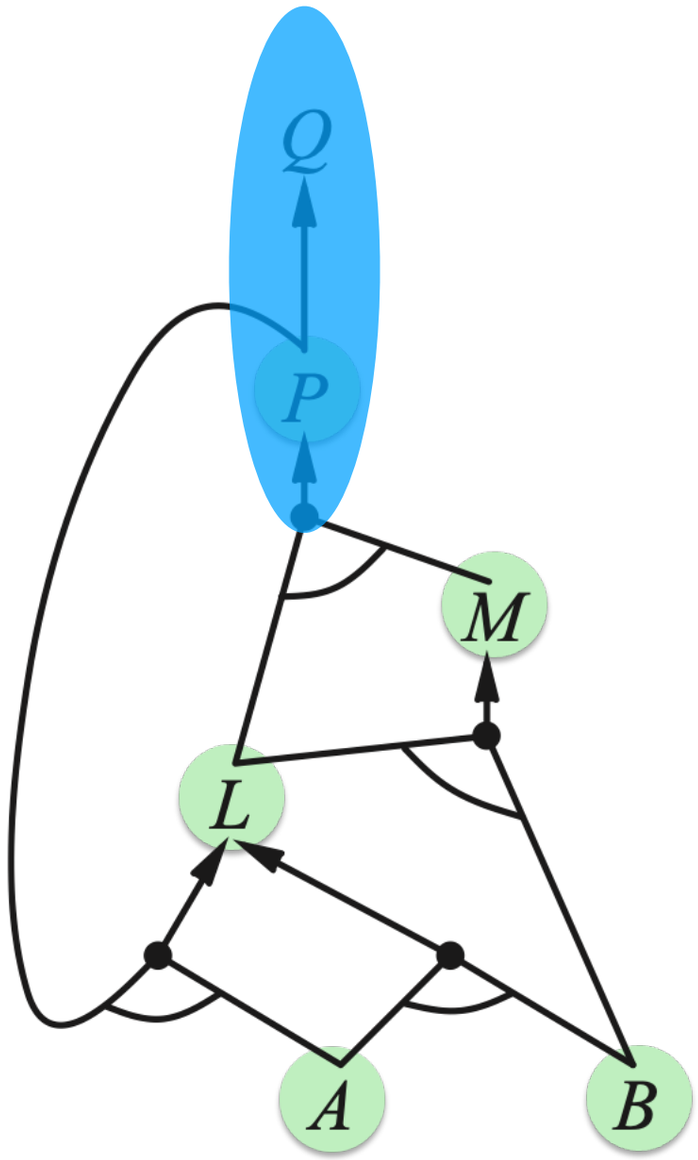
Slide courtesy: Stuart Russell & Sergey Levine

Backward Chaining Algorithm



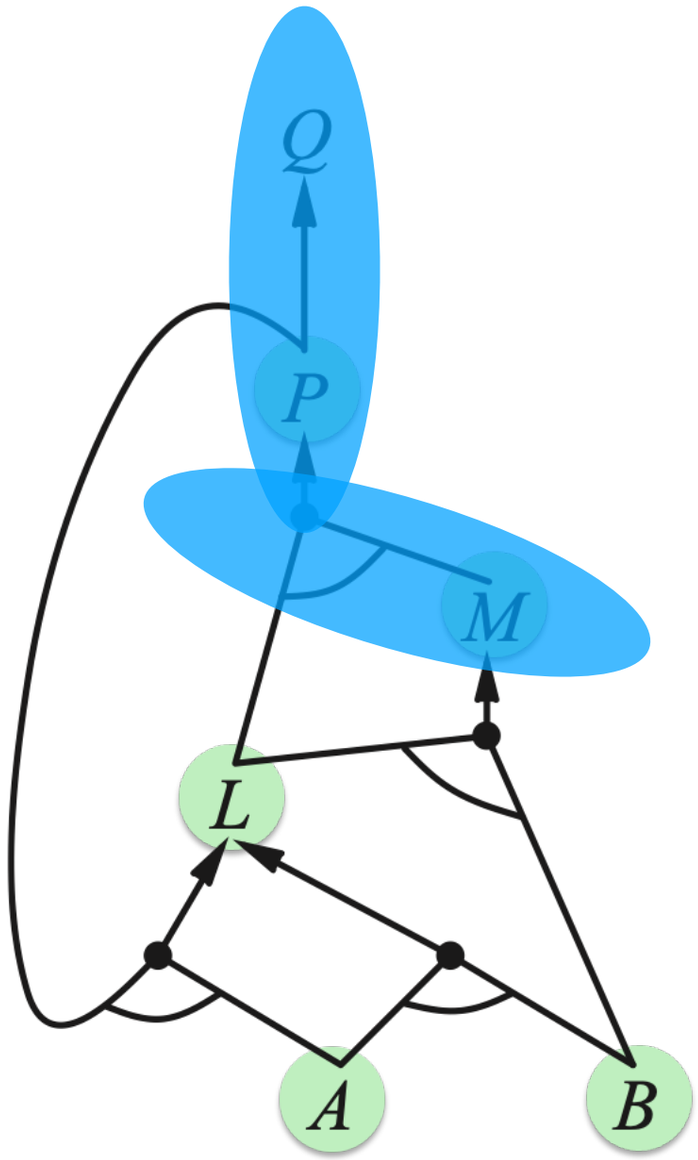
Tracing back from the target by depth-first search

Backward Chaining Algorithm



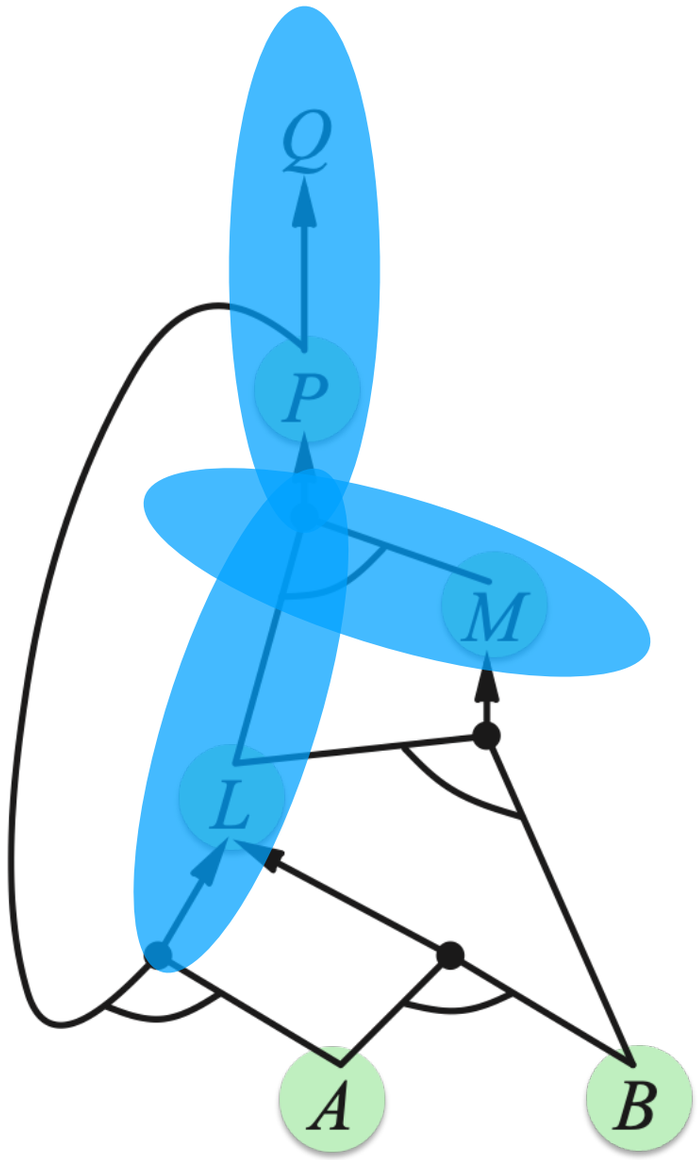
Tracing back from the target by depth-first search

Backward Chaining Algorithm



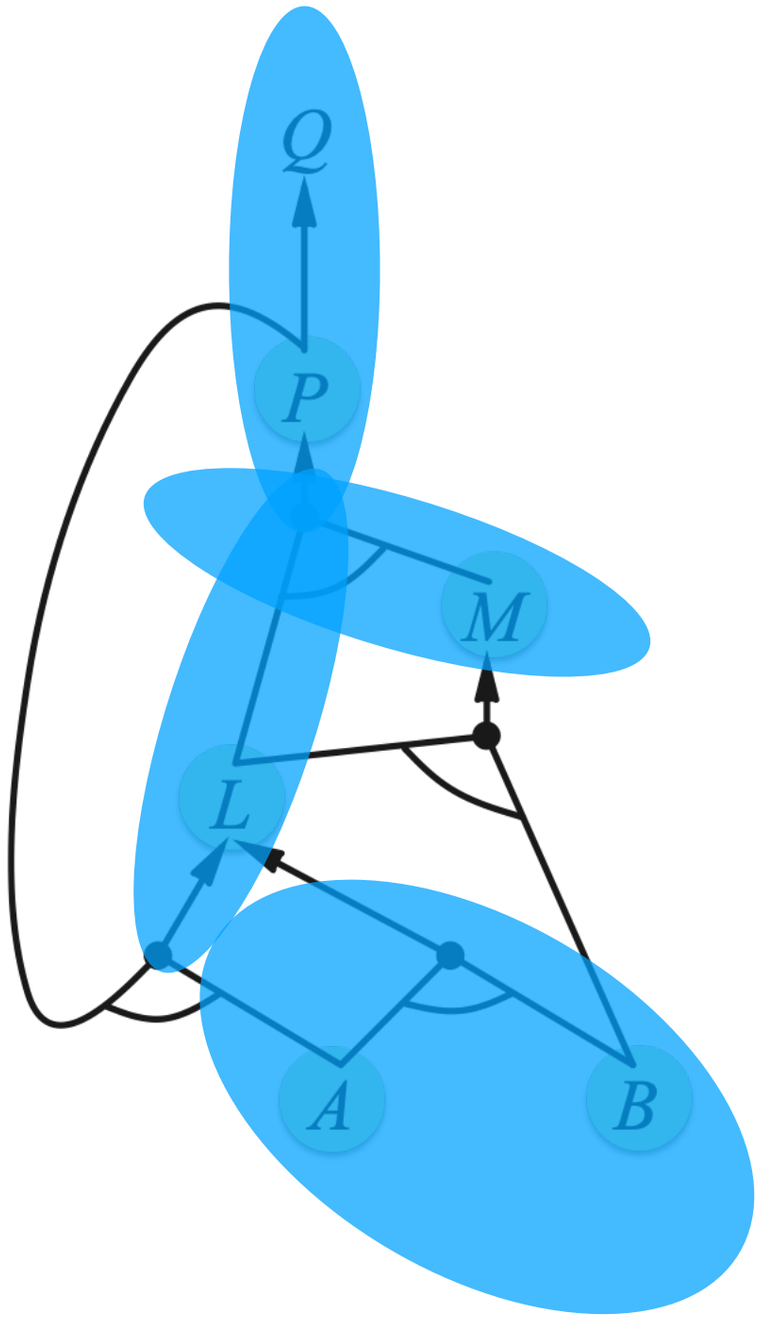
Tracing back from the target by depth-first search

Backward Chaining Algorithm



Tracing back from the target by depth-first search

Backward Chaining Algorithm



Tracing back from the target by depth-first search

Efficient SAT Solvers

- DPLL (Davis-Putnam-Logemann-Loveland) is the core of modern solvers
- Essentially a backtracking search over models with some extras:
  - ***Early termination***: stop if
    - all clauses are satisfied; e.g., $(A \vee B) \wedge (A \vee \neg C)$ is satisfied by {A=true}
    - any clause is falsified; e.g., $(A \vee B) \wedge (A \vee \neg C)$ is satisfied by {A=false,B=false}
  - ***Pure literals***: if all occurrences of a symbol in as-yet-unsatisfied clauses have the same sign, then give the symbol that value
    - E.g., A is pure and positive in $(A \vee B) \wedge (A \vee \neg C) \wedge (C \vee \neg B)$ so set it to true
  - ***Unit clauses***: if a clause is left with a single literal, set symbol to satisfy clause
    - E.g., if A=false, $(A \vee B) \wedge (A \vee \neg C)$ becomes $(\text{false} \vee B) \wedge (\text{false} \vee \neg C)$, i.e. $(B) \wedge (\neg C)$
    - Satisfying the unit clauses often leads to further propagation, new unit clauses, etc.

Efficient SAT Solvers

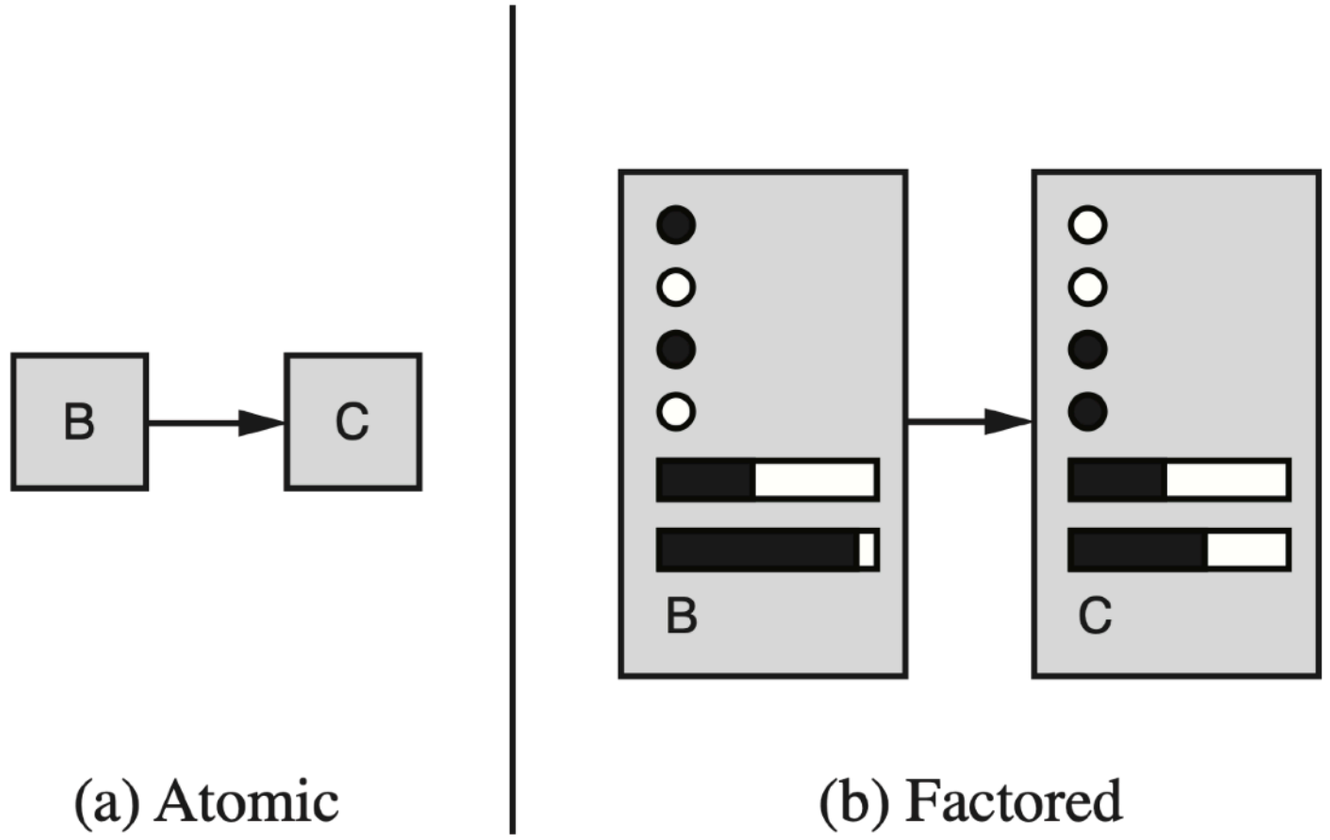
- DPLL (Davis-Putnam-Logemann-Loveland) is the core of modern solvers
- Essentially a backtracking search over models with some extras:
  - ***Early termination***: stop if
    - all clauses are satisfied; e.g., $(A \vee B) \wedge (A \vee \neg C)$ is satisfied by {A=true}
    - any clause is falsified; e.g., $(A \vee B) \wedge (A \vee \neg C)$ is satisfied by {A=false,B=false}
  - ***Pure literals***: if all occurrences of a symbol in as-yet-unsatisfied clauses have the same sign, then give the symbol that value
    - E.g., A is pure and positive in $(A \vee B) \wedge (A \vee \neg C) \wedge (C \vee \neg B)$ so set it to true
  - ***Unit clauses***: if a clause is left with a single literal, set symbol to satisfy clause
    - E.g., if A=false, $(A \vee B) \wedge (A \vee \neg C)$ becomes $(\text{false} \vee B) \wedge (\text{false} \vee \neg C)$, i.e. $(B) \wedge (\neg C)$
    - Satisfying the unit clauses often leads to further propagation, new unit clauses, etc.

Any modern SAT solvers can be used for logic inference!

Knowledge Reasoning: I

- Logic Reasoning
- Propositional Logic
  - Search-Based Inference
  - Resolution-Based Inference
  - Forward and Backward Chaining
- First-Order Logic
- Take-Home Messages

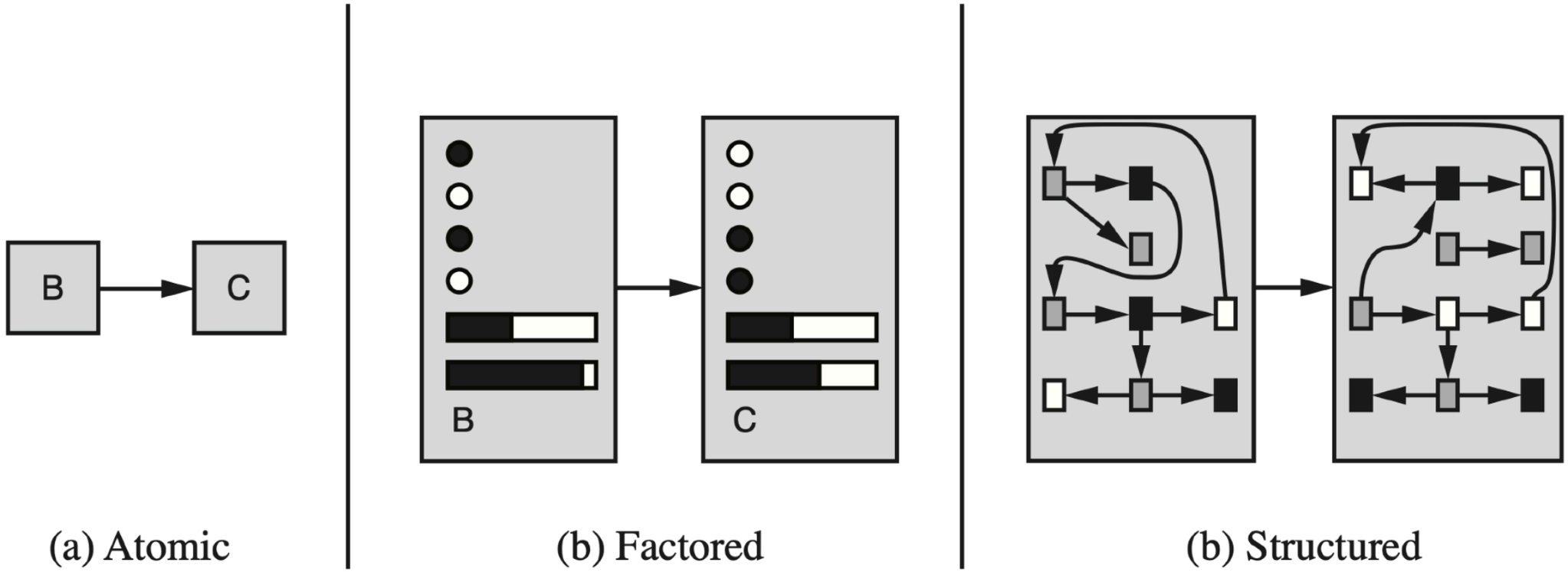
The Power of Propositional Logic



Propositional logic can represent factorized variables.
A big step forward to basic search strategies which can only represent the world with atomic (non-variable) representations.

Slide courtesy: Stuart Russell & Sergey Levine

The Limitation of Propositional Logic



(a) Atomic | (b) Factored | (b) Structured

Search, game-playing

CSPs, planning, propositional logic, Bayes nets, neural nets

First-order logic, databases, probabilistic programs

But propositional logic only models factorized variables, ignoring the relations among them.

student(ZJU, you)

Slide courtesy: Stuart Russell & Sergey Levine

First-Order Logic

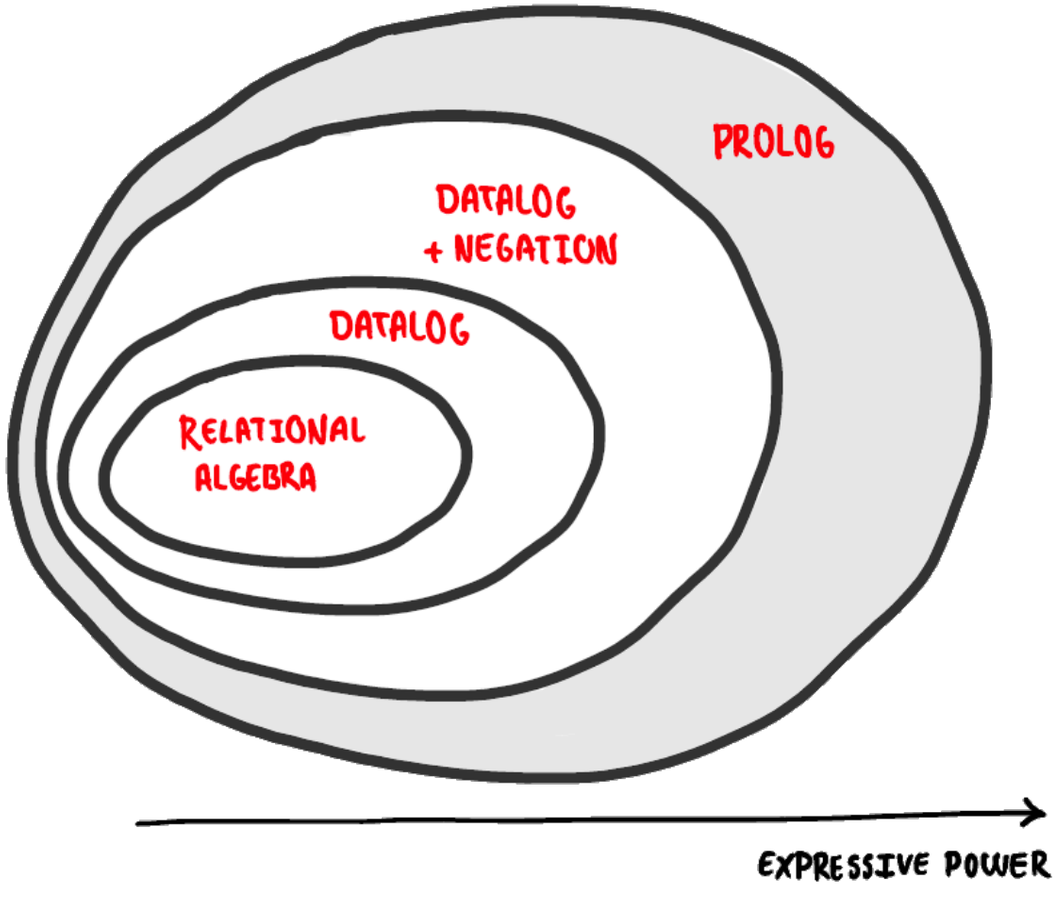
- The most significant improvements over propositional logic:
  - Including predicates: Student(ZJU, you)
  - Including functions: Grade(ZJU, you)

predicates output true/false,
functions output values.

- Including quantifiers: $\exists$, $\forall$
  - Rules of chess:
    - 100,000 pages in propositional logic
    - 1 page in first-order logic
  - Rules of pacman:
    - $\forall x,y,t\ At(x,y,t) \Leftrightarrow [At(x,y,t-1) \wedge \neg \exists\ u,v\ Reachable(x,y,u,v,Action(t-1))]\ \vee$
      $[\exists\ u,v\ At(u,v,t-1) \wedge Reachable(x,y,u,v,Action(t-1))]$

First-Order Logic Inference

- Propositionalization: reduce to propositional logic problem.
- Direct first-order logic approaches:
  - first-order forward and backward chaining



Knowledge Reasoning: I

- Logic Reasoning
- Propositional Logic
  - Search-Based Inference
  - Resolution-Based Inference
  - Forward and Backward Chaining
- First-Order Logic
- Take-Home Messages

Take-Home Messages

- Knowledge reasoning systems are used to build internal models of agents for modeling and representing the real world.
- Logic inference is the most classical method for knowledge reasoning, which deals with discrete and deterministic problems.
- Propositional logic characterize the system of factorized variables.
- First-order logic can further characterize the relationships among variables.
- The semantic grounding problem is one of the frontier topics of AI research.

Next class: probabilistic reasoning

Thanks for your attention!
Discussions?

Acknowledgement: Many materials in this lecture are taken from https://inst.eecs.berkeley.edu/~cs188/sp19/